Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Multiagentní systém umožňující agentům utkat se v jednoduché hře

Plzeň 2020

Jindřiška Reismüllerová

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jindřiška REISMÜLLEROVÁ**
Osobní číslo:	**A17B0337P**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informatika**
Téma práce:	**Multiagentní systém umožňující agentům utkat se v jednoduché hře**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Prostudujte metody návrhu a realizace multiagentních systémů, zejména pro účely implementace takového systému na podporu výuky předmětu KIV/ISW.
2. Navrhněte multiagentní systém umožňující agentům utkat se v jednoduché hře, ve které bude možné uplatnit techniky strojového učení.
3. Navržený systém implementujte.
4. Implementujte několik rozdílných pravidlově řízených agentů, na kterých systém otestujete.
5. Práci zdokumentujte a objektivně zhodnoťte dosažené výsledky.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Ing. Jakub Sido**
Nové technologie pro informační společnost

Datum zadání bakalářské práce: **7. října 2019**
Termín odevzdání bakalářské práce: **7. května 2020**

Doc. Dr. Ing. Vlasta Radová
děkanka

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 15. října 2019

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracovala samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 7. května 2020

Jindřiška Reismüllerová

Prohlášení o ochranných známkách

Názvy programových produktů, technologií, firem apod. použité v textu mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Poděkování

Ráda bych poděkovala Ing. Jakubu Sidovi za odborné vedení, cenné rady a pomoc při zpracování této práce. Dále bych ráda poděkovala Ing. Martinu Úblovi za konzultace k síťové části práce a v neposlední řadě Ing. Kamilu Ekšteinovi, Ph.D. za věcné poznámky k textu práce.

Abstract

The main goal of this bachelor thesis is to design and implement a generic client-server environment that would serve as a program for teaching multiagent systems. The subsequent application of this environment is to create a simple game and implement agents with different attributes, followed by a comparison of these implementations. This environment should be scalable and suitable for further development in the areas of game theory and artificial intelligence.

Abstrakt

Cílem této bakalářské práce je navrhnout a implementovat generické klient-server prostředí, jež by sloužilo jako program pro výuku multiagentních systémů. Dále pak použití tohoto prostředí umožní tvorbu jednoduché hry a implementaci agentů s různými vlastnostmi s následným porovnáním těchto implementací. Toto prostředí by mělo být snadno rozšiřitelné a vhodné pro další rozvoj v oblastech teorie her a umělé inteligence.

Obsah

1 Úvod **9**

2 Agentní systémy **10**
2.1 Agent . . . 11
2.1.1 Typy agentů . . . 11
2.2 Multiagentní systémy . . . 13
2.2.1 Centralizovaný multiagentní systém . . . 13
2.2.2 Decentralizovaný multiagentní systém . . . 13
2.2.3 Federovaný multiagentní systém . . . 14

3 Návrh systému **15**
3.1 Požadavky na aplikaci . . . 15
3.2 Dosavadní software . . . 16
3.2.1 Pogamut 3 . . . 16
3.3 Návrh hry . . . 17
3.3.1 Pravidla hry . . . 18
3.3.2 Parametry hry . . . 18
3.4 Zvolené programovací jazyky . . . 20
3.5 Struktura systému . . . 20
3.6 Moduly . . . 21
3.6.1 Modul síťové komunikace . . . 21
3.6.2 Modul herní logiky . . . 22
3.6.3 Modul agent . . . 22
3.6.4 Modul graf . . . 22
3.6.5 Modul tým . . . 22
3.6.6 Modul transportního skriptu . . . 22
3.6.7 Modul mediátor . . . 22
3.7 Časové uspořádání tahů . . . 23
3.8 Reprezentace dat . . . 24
3.9 Protokoly . . . 24
3.9.1 Síťový protokol . . . 25
3.9.2 Protokol meziagentní komunikace . . . 31
3.10 Synchronizace v rámci komunikace . . . 31

4 Implementace **33**
4.1 Vlákna . . . 33
4.2 Komponenty . . . 34
4.2.1 Server v C++ . . . 34
4.2.2 Python . . . 36
4.3 Transparentnost přenášených zpráv . . . 37
4.4 Implementovaní agenti . . . 38
4.4.1 Struktura agentů . . . 38
4.4.2 Pasivní agent . . . 39
4.4.3 Agent průzkumník . . . 39
4.4.4 Agent útočník . . . 39
4.4.5 Agent útočník–podvodník . . . 40
4.4.6 Agent obránce . . . 40
4.5 Vizualizace . . . 41

5 Dosažené výsledky **42**

6 Závěr **49**

A Uživatelská příručka **52**

1 Úvod

S multiagentními systémy se v dnešním světě setkáváme na každém kroku [21]. Používají se k řešení nejrůznějších problémů od aukcí přes plánování řešení katastrof a zpracování zdravotních záznamů, až po plánování vojenských misí. Multiagentní systémy řeší problémy, které nemají dopředu jednoznačně daný průběh a často je třeba přizpůsobovat chování změnám prostředí a umět na ně reagovat přímo za běhu. Agent tak musí být schopen reagovat na podněty z prostředí či od jiných agentů a zároveň na základě těchto podmětů upravit své budoucí chování. Jedná se tak o formu učení a z tohoto důvodu můžeme multiagentní systémy klasifikovat jako odvětví umělé inteligence.

Za poslední tři desetiletí došlo v oblasti multiagentních systémů k obrovskému rozmachu a v současné době se s nimi již běžně setkáváme v praxi [7]. To ovšem neznamená, že by již v této oblasti nebylo co vylepšovat a vyvíjet. Multiagentní systémy jsou velice komplexní a nabízejí spoustu různých možností jejich modifikace k řešení široké škály úloh. Proto je důležité pokračovat v jejich výzkumu.

Cílem této práce je vyvinout multiagentní prostředí, které bude sloužit zejména k výuce multiagentních systémů na Západočeské univerzitě v Plzni. Tento program bude implementovat jednoduchou grafovou hru, do které budou studenti moci doprogramovat vlastní agenty a utkat se s agenty jiného hráče. Do budoucna se také plánuje hru v určité formě zveřejnit na webu a moci tak od uživatelů sbírat data, podle kterých by se mohli za pomoci algoritmů strojového učení vyvinout inteligentní agenti.

2 Agentní systémy

Cílem této kapitoly je uvést čtenáře do problematiky agentních systémů a vysvětlit mu některé pojmy, které mu usnadní pochopení práce.

Koordinace

Koordinace mezi agenty je důležitá pro dosahování společných cílů. Může být buď centralizovaná nebo decentralizovaná. Centralizovanou rozumíme takovou koordinaci, kdy jeden agent řídí ostatní, dává jim příkazy co mají dělat, aby společně dosáhli daného cíle. Naopak decentralizovaná kooperace probíhá přímo mezi agenty.

Kooperace

U decentralizované koordinace, která se neřídí protokolem, se agenti musejí nějakým způsoben dohodnout, nalézt společně řešení konfliktů atd. Takové procesy nazýváme *kooperací*.

Negociace

V případě sociálních agentů (viz refkap:soc) je často potřeba ke kooperaci nebo řešení konfliktů *negociace*. Jedná se v podstatě o posloupnost zasílaných zpráv mezi agenty, která vede k vzájemné dohodě. Negociaci si můžeme představit jako proces, ve kterém si mezi sebou agenti navrhují řešení, dokud s nějakým všichni nesouhlasí.

Emergence

Jedná se o spontánní chování systému, které vzniká jako důsledek chování jednotlivých komponent (agentů) a interakcemi mezi nimi. Emergenci dělíme na slabou a silnou. V případě slabé emergence vzniká chování běžně neočekáváné, avšak při znalosti vlastností jednotlivých komponent a vazeb mezi nimi je možné toto chování odvodit a popsat. Naopak v případě silné emergence je chování systému nemožné odvodit z chování koponent a jejich vazeb a často tak představuje problém.

2.1 Agent

Pojem *agent* můžeme chápat jako samostatnou jednotku operující pouze ve vymezeném prostředí, která je schopna plnit určité úkoly bez většího zásahu člověka (správce systému), ten by měl plnit pouze funkci dohledu. Pokud bychom na agenta nahlíželi z programátorského hlediska, jedná se o entitu podobnou objektu, avšak je na něj nahlíženo s ještě větší abstrakcí. Porovnání objektu a agenta je popsáno následující tabulkou 2.1.

Objekt	**Agent**
Zapouzdřuje proměnné a metody	Zapouzdřuje chování
Nutná synchronizace vláken	Agenti nejsou navzájem nezávislí
Uspořádání objektů prostřednictvím dědičnosti a kompozice	Uspořádání agentů v organizacích
Komunikuje voláním metod (posíláním zpráv)	Komunikuje prostřednictvím vyšších komunikačních jazyků
Prostředí (kromě kompilačního) nehraje žádnou roli	Významnou roli sehrává prostředí

Tabulka 2.1: Srovnání abstrakce objektů a agentů; podle [17]

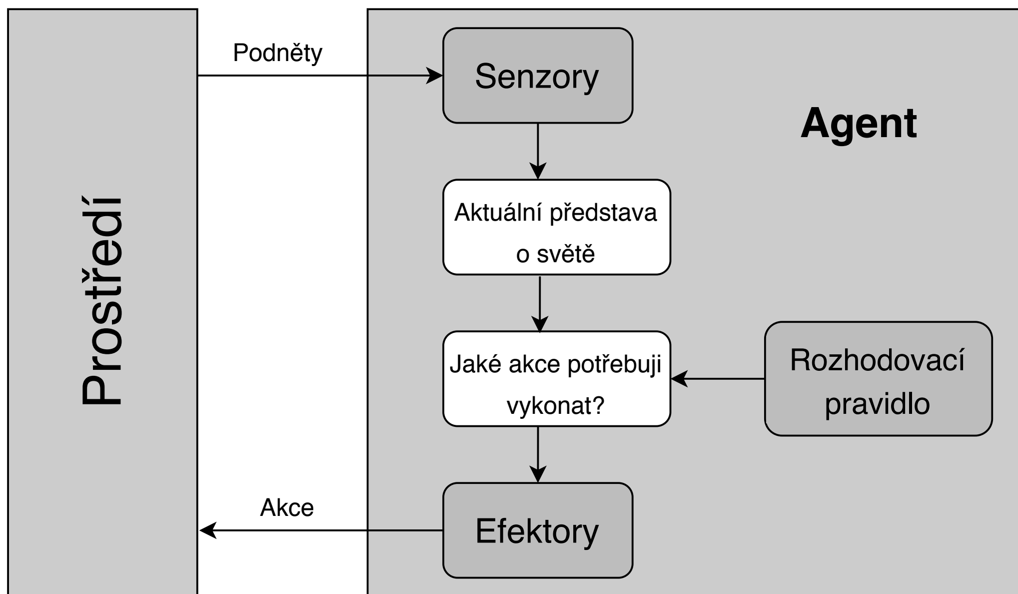
2.1.1 Typy agentů

Aleš Kubík v [17] rozlišuje čtyři základní typy agentů: reaktivní, deliberativní, sociální a hybridní. V této části budou jednotlivé druhy krátce popsány.

Reaktivní agent je nejjednodušším typem agenta. Jeho základním rysem je, že si vnitřně neuchovává žádnou reprezentaci okolního světa a výsledek jeho rozhodování se zakládá na kombinaci jeho aktuálního stavu a vjemů z prostředí. Speciálním případem reaktivního agenta je čistě reaktivní agent, v některých literaturách také označovaný jako „reflexní agent“. Od reaktivního se liší tím, že nedisponuje žádnou interní pamětí. Neuchovává si tudíž

žádný stav a jeho rozhodování se tak odvíjí čistě od podnětů z prostředí.

Deliberativní agent se od reaktivního liší tím, že si vytváří a uchovává model prostředí, ve kterém se pohybuje a může tak plánovat své akce dopředu. Reálný svět se snaží pospat značkami (symboly), které si vytváří porovnáváním reálných objektů se známými vzory, a vytváří si tak symbolickou reprezentaci světa. Pro deliberativní agenty se někdy můžeme setkat i s pojmenováním „intencionální“ nebo též „uvažující“. Vztah deliberativního agenta s prostředním je popsán obrázkem 2.1.



Obrázek 2.1: Deliberativní agent ve vztahu k prostředí; převzato z [2]

Sociální agent disponuje kromě vlastního modelu světa také informacemi o ostatních agentech a možností s nimi konunikovat. Je tak schopen například negociace. To je pro agenta důležité zejména kvůli omezenosti vlastních zdrojů. Sociální agenti se mohou účastnit aukcí a celkově se zdroji obchodovat mezi ostatními agenty. Agent si ukládá informace o proběhlých transakcích a později je může použít při rozhodování (například má-li na výběr dva agenty, s kterými může obchodovat a s oběma už dříve obchodoval, zvolí toho, se kterým pro něj bude obchod pravděpodobně výhodnější).

U **hybridního agenta** můžeme najít vlastnosti všech výše popsaných. Zahrnuje jak reaktivní komponenty pro reakce na podněty z prostředí, tak deliberativní prvky a v neposlední řade i sociální model pro komunikaci

s ostatními agenty.

2.2 Multiagentní systémy

Multiagentním systémem rozumíme systém, skládající se z vícero agentů, již jsou spolu schopni interagovat. Jednotliví agenti přitom nemají dostatek dat a potřebných zdrojů, aby mohli samostatně dosáhnout svého, či společného cíle. Jsou tudíž nuceni spolupracovat s ostatními agenty. Multiagentní systémy lze rozdělit na centralizované a decentralizované (viz 2.2.1 a 2.2.2).

Agenti se mezi sebou dorozumívají pomocí komunikačních jazyků. Již od 90. let se rozvíjí dva jazyky, a to *KQML* (*Knowledge Query and Manipulation Language*) [9] a *ACL* (*Agent Communication Language*) [15] organizace FIPA, který je zpravidla označován jako *FIPA-ACL*.

2.2.1 Centralizovaný multiagentní systém

Centralizovaný multiagentní systém disponuje jedním nadřízeným agentem, jenž řídí komunikaci mezi jednotlivými agenty a dohlíží na plnění jejich úkolů a tím na dosažení jejich i společných cílů. Veškerá komunikace tak plyne pouze přes řídícího agenta, z čehož plyne složitá implementace centrálního agenta. Speciálním případem centralizovaných systémů je takzvaný hierarchický multiagentní systém, kde existuje více úrovní řídících agentů, přičemž každý agent může komunikovat pouze s jedním agentem vyšší úrovně. Komunikaci v této architektuře si můžeme představit jako strom, jehož listy jsou výkonní agenti.

2.2.2 Decentralizovaný multiagentní systém

Decentralizovaný multiagentní systém na druhou stranu neobsahuje žádnou řídící autoritu. Agenti si vytvářejí plány samostatně a veškerá komunikace mezi nimi je přímá. Implementace decentralizovaných systémů může být často složitá, například z důvodu negociací, kde se spolu agenti musejí dohodnout, narozdíl od centralizovaných systémů, kde o výsledku snadno rozhodne řídící agent.

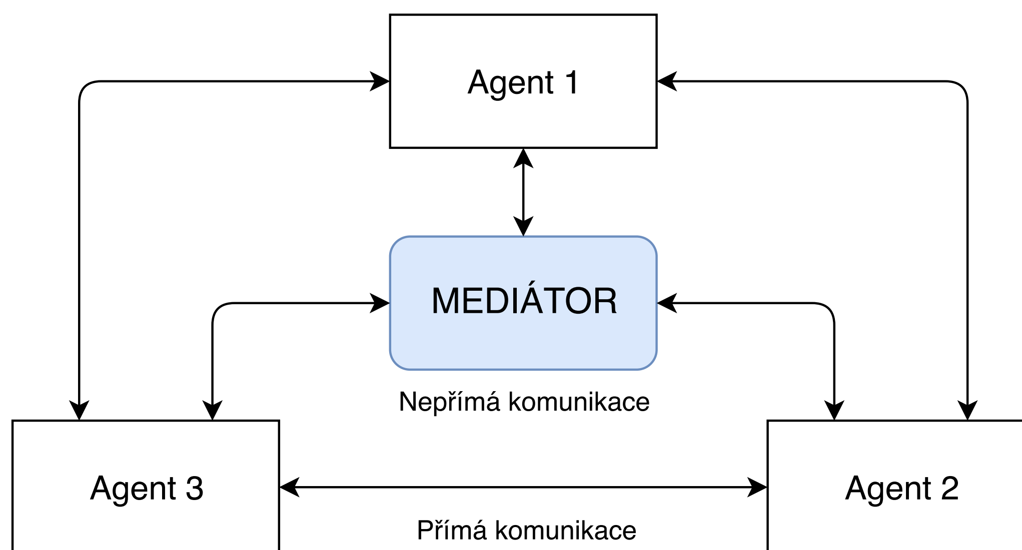
Decentralizované systémy mohou být buď statické, nebo dynamické.

V případě statických jsou pevně dané vztahy mezi jednotlivými agenty a výpadek může snadno způsobit problémy. Také přidání nového agenta není v případě statických systémů snadné a v některých případech ani možné. Výhodou statických systémů je však snažší implementace.

Naopak přidání nebo odebrání agenta z dynamického systému vyžaduje minimální náklady. Vazby mezi agenty jsou uvolněnější, což ovšem vede ke složitější implementaci a nejistým výsledkům.

2.2.3 Federovaný multiagentní systém

Federovaný systém se vyznačuje přítomností zprostředkovatelského agenta (tzv. *mediátor*) využívaného pro nepřímou komunikaci agentů. Agenti však nejsou povinni využívat mediátor ke komunikaci a můžou mezi sebou komunikovat i přímo. Nevylučuje se ani přítomnost více mediátorů. Struktura federovaného systému je zobrazena na obrázku 2.2.



Obrázek 2.2: Architektura federovaného multiagentního systému; podle [17]

Tato architektura je zde popsána samostatně, neboť ji nelze zařadit do centralizovaných systémů, protože se agenti rozhodují samostatně, neexistuje tedy žádná nadřízená autorita. Nelze ji však považovat ani za zcela decentralizovanou, jelikož komunikace mezi agenty neprobíhá pouze přímo.

3 Návrh systému

Tato kapitola se bude zabývat zejména celkovým rozvržením systému. Budou zde popsány jednotlivé moduly, protokoly a formáty přenášených zpráv.

3.1 Požadavky na aplikaci

Se záměrem využití výsledného produktu v předmětu KIV/ISW a zadáním práce je třeba splnění následujících požadavků:

- **Dobře zdokumentované a jednoduché programátorské rozhraní** – S programem musí být snadné pracovat, uživatelé by se měli za pomoci dokumentace rychle vyznat v programátorském rozhraní a být schopni bez větších obtíží naimplementovat inteligentního agenta a připojit ho do hry. Rovněž zbytek programu musí být dobře zdokumentovaný z důvodu rožšiřitelnosti v rámci předmětu KIV/ISW.

- **Izolace herní logiky od implementace agentů** – O herní logiku by se měl starat server, zatímco agenti mu budou pouze posílat požadavky na akce, které by rádi vykonali. Veškerá ošetření validity požadavků a jejich samotné vykonání by ale mělo proběhnout na serveru. Server posílá každému agentovi jen část grafu hry, která má být pro agenta v daný okamžik viditelná.

- **Neexistence univerzální výherní strategie** – Nesmí být možné napsat jednoho pravidlově řízeného agenta, který by byl schopný vyhrát hru za všech okolností. Cílem je donutit agenty přizpůsobovat se aktuálnímu prostředí a podněcovat je ke spolupráci pro dosažení společného cíle.

- **Podpora simulace** – Možnost spustit aplikaci v módu simulace. Klíčovou vlastností simulace je časová komprese, díky které je možné zrychlit průběh hry na nejkratší možný časový úsek [3].

- **Přenositelnost kódu** – Kód musí být přenositelný mezi operačními systémy MS Windows a GNU/Linux.

- **Robustnost** – Všechny uživatelské vstupy a přenášené zprávy musí být správně ošetřeny. Zprávy nevyhovující protokolu povedou k přerušení spojení s protistranou.

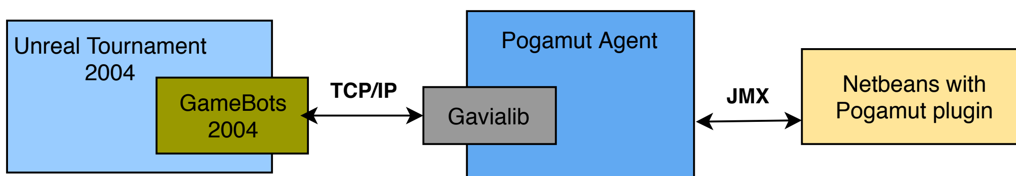
Jako další požadavek, jehož nesplnění však není v rozporu se zadáním, byla uvedena vizualizace a webové uživatelské rozhraní, za pomoci něhož by se mohli utkat reální hráči buď proti sobě, nebo proti týmu agentů. Každý hráč by pak představoval jeden tým a se svými agenty by pohyboval jako s figurkami. Tento požadavek byl uveden s vyhlídkou budoucího nasazení na web a strojového učení agentů z výsledků jejich her s lidmi.

3.2 Dosavadní software

V současné době se multiagentní systémy na Západočeské univrzitě v Plzni vyučují za pomoci aplikace *Pogamut 3* [11], proto bude v této kapitole podrobněji popsána.

3.2.1 Pogamut 3

Pogamut 3 je middleware, který poskytuje jednotné rozhraní pro tvorbu agentů. Samotná hra musí implementovat rozhraní pro ovládání těchto agentů, nejčastěji je to protokol 'GameBots 2004'. Na straně Pogamutu je pak knihovna, která rovněž implementuje tento protokol, zpravidla je to knihovna Gavialib. Protokol GameBots operuje na prezentační vrstvě ISO/OSI modelu [14]. Popsaným způsobem je odstíněna síťová komunikace od logiky agenta. Nejčastěji se Pogamut aplikuje na hru Unreal Tournament. Je ale také aplikovatelný například na hru Minecraft a jiné. Schéma 3.1 popisuje architekturu rozhraní Pogamut.



Obrázek 3.1: Architektura rozhraní Pogamut; převzato z [11]

Ve vztahu k požadavkům stanoveným v kapitole 6.1 bude uvedeno, do jaké míry jim Pogamut vyhovuje.

- **Dobře zdokumentované a jednoduché programátorské rozhraní** – Rozhraní je dobře zdokumentované vzhledem k dlouholetému vývoji. Jelikož je Pogamut aplikovatelný na širokou škálu her, je jeho implementace značně komplexní. Při vývoji nového řešení bude tedy kladen důraz zejména na přímočarost a rozšiřitelnost rozhraní.
- **Izolace herní logiky od implementace agentů** – Tento bod Pogamut splňuje v plné míře. Bude tedy možné se jím v tomto bodě inspirovat.
- **Neexistence univerzální výherní strategie** - Pogamut je pouze knihovna pro zprostředkování práce s agenty. Splnění tohoto bodu je tudíž závislé na logice hry, která Pogamut používá.
- **Podpora simulace** – Pogamut je stavěn na běh v reálném čase a nepodporuje časovou kompresi. Toto je hlavní nedostatek, jelikož kvůli tomu není umožněno v krátkém časovém úseku odehrát hru, což je stěžejní pro oblast umělé inteligence a strojového učení.
- **Přenositelnost kódu** – Pogamut je napsán v programovacím jazyce Java, je tedy přenositelný.
- **Robustnost** – Vzhledem k velké uživatelské základně lze předpokládat, že je aplikace dostatečně robustní.

Cílem nové aplikace bude vytvořit multiagentní systém s rozhraním pomocí kterého spolu budou agenti moci komunikovat. V případě Pogamutu není hlavním cílem vytváření multiagentního prostředí, nýbrž individuálních agentů.

Pogamut je distribuován pod licencí GNU GPLv3 [12]. Protokol GameBots pod vlastní, proprietární licencí. Kombinace licenčních podmínek těchto licencí může být problematická pro další vývoj a aplikace.

3.3 Návrh hry

Tato kapitola má za cíl seznámit čtenáře s pravidly navržené hry a parametry, se kterými je tuto hru možné spustit a tím ovlivnit počáteční podmínky a následný průběh hry.

3.3.1 Pravidla hry

Jak již bylo řečeno, jedná se o grafovou hru v reálném čase. V uzlech grafu se mohou nacházet agenti, kredity, nebo je může právě jeden z týmů vlastnit. Na začátku hry jsou proti sobě postaveny 2 různé týmy agentů. Při náhodném rozmístění hráčů je možné přidat do hry více týmů a to i během již započaté hry. Takto je možné přidávat týmy dokud existují uzly, které nevlastní ani jeden z týmů. Každý tým je složen z agentů, jejichž úkolem je zabrat uzly protivníkova týmu.

Ve hře se také nachází uzly, jež generují kredity, tyto uzly budou dále označovány jako „generující“. Každý tým vlastní po přidání do hry právě jeden generující uzel.

Hra je řízena jednotkovými změnami času hry, tzv. *tiky*, kdy při každém tiku smí agent provést nanejvýš jednu akci. Po určitém počtu tiků pak všechny generující uzly vygenerují ve stejný čas jeden kredit. Pokud chce agent zabrat nějaký uzel, který nikomu nepatří, musí do něj položit alespoň jeden kredit. Pokud je takovýto uzel generující, všechny doposud vygenerované kredity v uzlu zůstávají. V případě, že chce agent sebrat uzel protivníka, musí do něj položit alespoň o jeden kredit víc, než se v něm právě nachází. Pokud do cizího uzlu položí menší počet kreditů, uzel zůstane protivníkovi, ale počet kreditů v něm se zmenší právě o počet položených. Agent tak může uzel zabrat opakovaným pokládáním kreditů. Kredity může agent sbírat pouze ve svých vlastních uzlech. Maximální počet kreditů, které může agent najednou přenášet, je omezen.

3.3.2 Parametry hry

Hra se dá spouštět s různými parmetry. Tím je zaručeno, že neexistuje jedna unikátní výherní strategie, ale agenti se musí přizpůsobovat nastaveným podmínkám, v nichž může hrát roli i náhoda. Existence unikátní výherní strategie je nežádoucí, protože od agentů chceme, aby se uměli přizpůsobit aktuální situaci a mohli se učit ze svých chyb, nikoli aby bylo možné naprogramovat každý tah agenta ještě před začátkem hry. Parametry, s kterými se dá hra spouštět, budou popsány níže.

Podmínka výhry

Hra může běžet v módu *Take all*, kdy tým prohraje až poté, co mu ostatní týmy seberou všechny jeho uzly. Druhý mód je *Take home*, kdy tým prohrává, pokud mu jiný tým sebere uzel, který mu byl přiřazen při vstoupení do hry, tedy domovský. V tomto případě se všechny jeho ostatní uzly nadále

chovají jako by mu patřily a pokud zbývají ve hře ještě alespoň dva týmy, hra pokračuje dál.

Vstup na cizí uzel

Při vytváření hry se může hráč rozhodnout, zda chce, aby agent nesoucí kredity byl penalizován za vstup na cizí uzel. Tento parametr se zpravidla zapíná v módu *Take home*, kde se pak hráčům nabízí strategie zabrat si uzly sousedící bezprostředně se svým domovským uzlem, aby do něj soupeř nikdy nemohl přinést plný počet kreditů. Avšak i v módu *Take all* může vést penalizace při vstupu na cizí uzel ke strategiím, kdy je protivník nucen například volit jiné cesty do generujících uzlů.

Start v náhodném uzlu

Začátek v náhodném uzlu je nutný při hře více než dvou týmů. Role náhody pak může některé týmy velmi zvýhodnit, jiné naopak. Proto byla přidána možnost, kdy se proti sobě utkají dva týmy se začátkem v uzlu s nejvyšším a nejnižším identifikačním číslem.

Limit agenta

Jako další parametr vytvářené hry je možné zadat maximální počet kreditů, jež může agent najednou přenášet. Tento parametr může velmi ovlivnit délku hry.

Generující uzly

Pro vývoj hry je také důležité, kolik tiků proběhne, než generující uzly vygenerují další kredity. Při generování kreditů po malém počtu tiků může hra trvat velice dlouho, jelikož agenti nebudou stíhat kredity rychle přenášet vzhledem k tomu, že každá akce agenta stojí jeden tik. Naopak při velkém počtu tiků na vygenerování jednoho kreditu se může stát, že jeden hráč porazí druhého během několika málo tiků.

Viditelnost agentů

Viditelnost agentů udává, jak daleko agent vidí. Jelikož je to hra grafová, rozumí se tímto parametrem, že má agent informace o uzlech se vzdáleností menší nebo rovnou hodnotě tohoto parametru. Čím je tato hodnota menší, tím víc je agent nucen ke komunikaci s ostatními agenty v týmu.

3.4 Zvolené programovací jazyky

Pro implementaci programu bylo využito tří programovacích jazyků.

Server, představující samotné multiagentní prostředí, je napsaný v C++. Tento jazyk byl zvolen z důvodu rychlosti. Server musí jednak vykonávat obsluhu síťového spojení s klienty a s tím související komunikaci, a na druhé straně zajišťuje běh her, a to jak v reálném čase, tak i jako simulaci, a to i v několika vláknech. Rychlost tedy byla stěžejním kritériem pro výběr jazyka pro server.

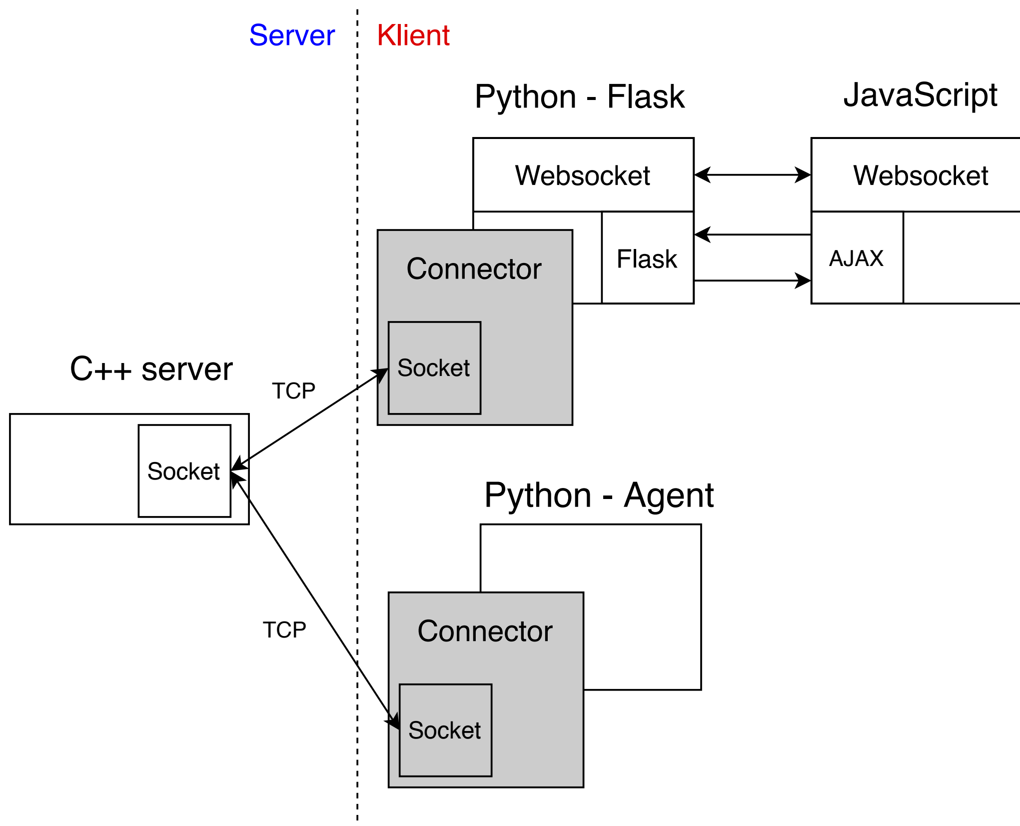
Pro klienta pak bylo nutné zvolit takový programovací jazyk, ve kterém je možné pracovat s objekty, s formátem JSON, který je dostatečně rozšířený a je možné uskutečnit propojení s webovým uživatelským rozhraním pro následné nasazení na web. Z těchto důvodů byl nakonec zvolen programovací jazyk *Python*, který tyto požadavky splňuje. Požadované propojení umí zajistit jeho knihovna *Flask*.

Pro vizualizaci byla zvolená kombinace jazyků HTML, JavaScript a CSS z již uvedeného důvodu, jímž je nasazení na web. Rozvržení systému je zobrazeno na obrázku 3.2.

3.5 Struktura systému

Jedním z hlavních poždavků je izolace herní logiky od implementace agentů. Z tohoto požadavku bude vycházet návrh celého systému. Herní logiku bude obstarávat server napsaný v C++, který bude od jednotlivých skriptů agentů v Pythonu oddělen síťovým rozhraním. Bude tak možné spustit server na samostatném výkonném zařízení, zatímco agenty bude možné spouštět na libovolných síťově dosažitelných výpočetních uzlech.

Pro komunikaci se serverem bude vytvořena samostatná knihovna, kterou bude využívat jak skript agenta, tak transportní skript pro komunikaci s webovým rozhraním napsaným v jazyce JavaScript. Knihovna tedy musí mít univerzální chování - nedělá rozdíl mezi agentem a lidským hráčem. Poslední částí systému je webové rozhraní, které musí být schopno zasílat a přijímat jak asynchronní zprávy (reakce na vstupy uživatele), tak synchronní (aktualizace stavu hry s každým tikem).



Obrázek 3.2: Schéma navržené struktury systému

3.6 Moduly

V této kapitole budou navrženy jednotlivé moduly, z nichž bude celý systém složen.

3.6.1 Modul síťové komunikace

Tento modul bude implementovat níže definovaný aplikační protokol se všemi náležitostmi. Modul síťové komunikace bude udržovat relaci, bude se starat o reprezentaci dat v rámci síťové komunikace a bude implementovat aplikační protokol. Jeho funkce odpovídá třem vrchním vrstvám ISO/OSI modelu (relační, prezentační a aplikační) [14]. Modul musí korektně rozpoznat chybné zprávy a reagovat na ně. Rovněž musí zajišťovat transparentnost přenosu dat.

3.6.2 Modul herní logiky

Modul herní logiky bude obstarávat validaci a provedení tahů agentů, generování kreditů a vyhodnocování konce hry. Bude se starat o herní tiky a průběh hry dle zadaných parametrů. Herní logika musí rozpoznat, zda hra probíhá v režimu simulace či nikoli a podle toho komprimovat čas tiků. Modul bude zajišťovat správné časové uspořádání tahů a jejich vykonání ve správný moment (bude popsáno blíže v kapitole 6.5).

3.6.3 Modul agent

Jedná se o modul podřízený modulu herní logiky. Entita agenta představuje právě jednoho agenta patřícího právě jednomu týmu. Agent se musí umět pohybovat po grafu. V jeden okamžik hry se bude nacházet v právě jednom uzlu. Musí být schopný nést, sbírat a pokládat kredity. Počet kreditů které nese v jeden okamžik, nesmí přesahovat nastavený limit agenta. Agent musí být schopen komunikovat s ostatními agenty v týmu pomocí mediátoru. Bude tedy třeba vytvořit federovaný multiagentní systém.

3.6.4 Modul graf

Graf se bude skládat z uzlů a hran. Uzel bude moci být generující s určitou pravděpodobností. Každý uzel bude patřit nejvýše jednomu z hráčů a bude možné jeho vlastnictví v průběhu hry měnit.

3.6.5 Modul tým

Tým se bude skládat z jednotlivých agentů. Každý z týmů bude vlastnit na začátku hry jeden generující uzel. Všichni agenti v týmu budou mít společný cíl. Pokud se bude jednat o interaktivní hru na webu, bude tým řízen právě jedním hráčem.

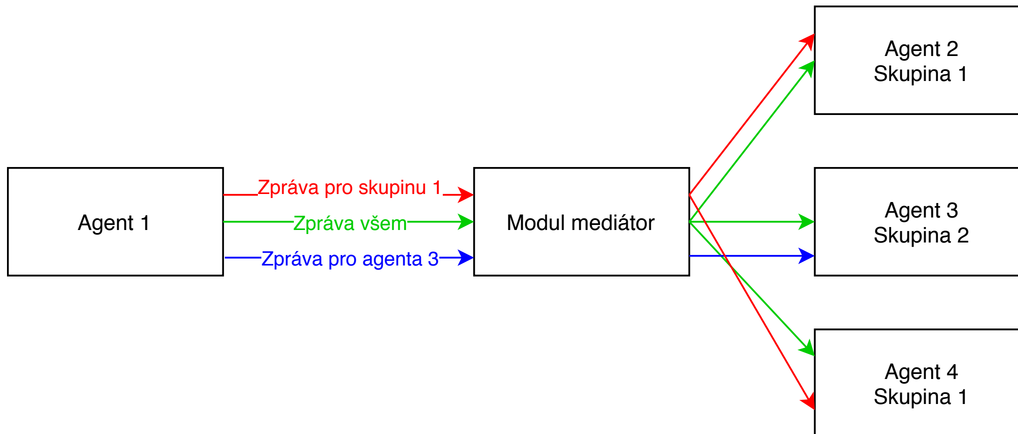
3.6.6 Modul transportního skriptu

Tento modul bude zajišťovat přenos zpráv mezi webovým rozhraním a serverem v C++. Bude muset přenášet jak zprávy asynchronní – reakce na vstupy uživatele, tak i synchronní – aktualizace stavu hry v rámci pravidelných tiků.

3.6.7 Modul mediátor

Modul mediátor bude zprostředkovávat komunikaci mezi jednotlivými agenty v týmu. Bude umožňovat zasílání zpráv respektujících meziagentní komuni-

kační protokol. Navíc bude možnost zasílat zprávy vlastního formátu v textové podobě, jejichž interpretace bude záviset čistě na programátorovi daného týmu agentů. Zprávy bude možné zasílat konkrétní podmnožině agentů v týmu. Možnosti rozlišení podmnožiny agentů jsou znázorněny na obrázku 3.3.



Obrázek 3.3: Možnosti zasílání zpráv podmnožinám agentů

3.7 Časové uspořádání tahů

Pro správný průběh hry bude důležité vykonávat jednotlivé tahy v určitém pořadí. Mezi tahy v tomto případě budeme počítat vygenerování kreditu generujícím uzlem, přesun agenta do jiného uzlu, položení a sebrání kreditů agentem. Bude třeba vyřešit jak pořadí tahů více agentů ve stejném uzlu, tak zaslání serveru více tahů jednoho agenta najednou.

Tahy se budou z důvodu logiky hry vykonávat v následujícím pořadí. Nejdříve se provede přesunutí všech agentů do požadovaných uzlů. Následně agenti provedou položení kreditů. Pokud by totiž proběhlo nejdřív vygenerování kreditů, agent by nezabral uzel položením o jeden kredit víc, než se v něm právě nachází, jak by to mělo být podle pravidel, ale musel by položit o dva víc. Pokládání kreditů může vést k změně vlastníka uzlu, je proto potřeba ho provést před sbíráním. Po pokládání bude následovat generování kreditů generujícími uzly. Nakonec bude provedeno sbírání kreditů.

V případě, že bude chtít v jednom uzlu ve stejný čas sebrat kredity více agentů, budou kredity v uzlu přiřazovány agentům cyklicky po jednom, do splnění požadavků agentů, nebo vyčerpání kreditů v uzlu.

Pro případ, kdy agent zašle v jeden okamžik více požadavků, bude vytvořen mechanizmus, který bude každé akci přiřazovat číslo. Toto číslo bude vyjadřovat, za kolik tiků se má daná akce provést. Při každém tiku bude číslo snižováno. Daný tah se provede, až toto číslo dosáhne hodnoty 0.

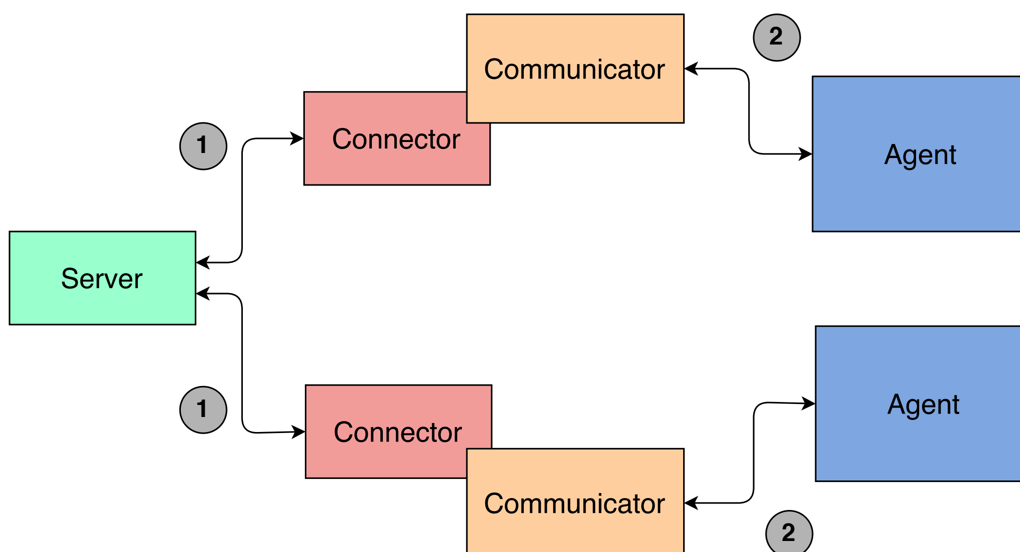
3.8 Reprezentace dat

Pro reprezentaci všech složitějších dat v rámci síťového přenosu bude použit formát JSON [4]. Bude použit z důvodu jednoduchosti a minimálního množství režijních dat. Další jeho výhodou je, že ze své definice zajišťuje transparentnost přenosu dat. Rovněž má podporu v široké škále programovacích jazyků.

Z dalších formátů se nabízel buď vlastní formát nebo formát XML [5]. Vytvoření vlastního formátu by znamenalo vymyslet reprezentaci každého z datových typů a bylo by nutné doimplementovat podporu všem použitým jazykům. Jako další varianta se nabízel formát XML, který ale nebyl použit kvůli většímu množství režijních dat, než je tomu u formátu JSON. Potenciál formátu XML by v této práci nebyl využit.

3.9 Protokoly

V této části budou popsány navržené protokoly pro komunikaci mezi entitami celého systému. Nejdřív bude popsán protokol pro komunikaci agenta se serverem. Následně pak protokol komunikace mezi agenty za pomoci navrženého rozhraní. Zasazení do kontextu je znázorněno na obrázku 3.4, kde agent zasílá zprávu, kterou chce poslat komunikačnímu rozhraní (na obrázku číslo 2). Tato zpráva je následně v modulu *Connector* zabalena a poslána na server, který jí rozešle příslušné podmnožině agentů.



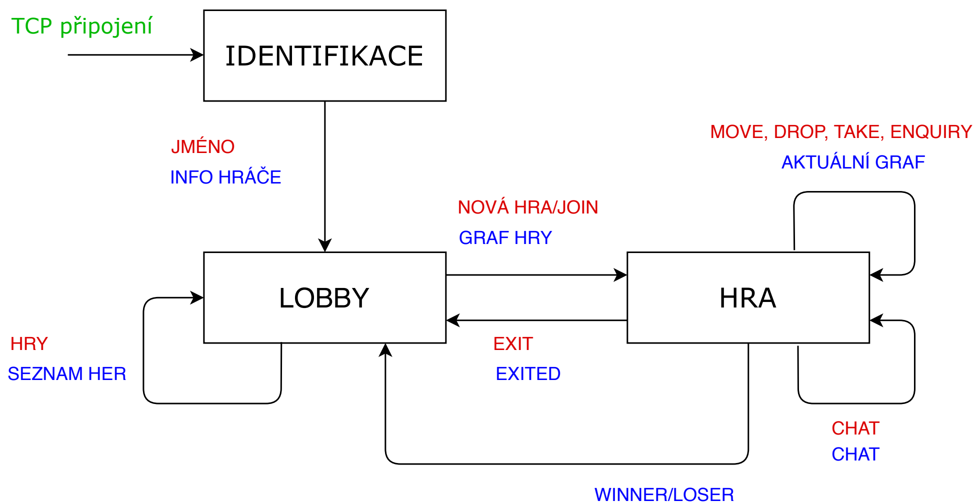
Obrázek 3.4: Protokoly v kontextu systému; číslem 1 je označen síťový protokol, číslem 2 protokol meziagentní komunikace.

3.9.1 Síťový protokol

Síťový protokol bude přenášet jak jednotlivé požadavky klientů (agentů) a příslušné odpovědi, tak i pravidelně aktualizované stavy hry. Zdrojem požadavku je vždy klient. Na každý požadavek server odpovídá vždy právě jednou zprávou. Zprávu s aktualizovaným stavem hry zasílá vždy server klientovi, a to ve vymezených časových intervalech bez předchozího požadavku klienta. Na tuto zprávu klient nemusí odpovídat.

Na protokol nebyly kladeny požadavky na bezpečnost, proto jsou zprávy v otevřené nešifrované podobě zasílány ve formátu JSON. Pro přenos zpráv byl zvolen transportní protokol TCP [20], protože je nutné zaručit spolehlivost přenosu a zachování pořadí zpráv.

S ohledem na výše uvedené požadavky byl navržen následující protokol. Přenos zpráv navrženého protokolu je znázorněn schématem 3.5 a přenášené zprávy jsou popsány v tabulce 3.1



Obrázek 3.5: Schéma přenášených zpráv; modře jsou znázorněny zprávy od serveru ke klientovi, červeně v opačném směru

Následuje seznam vysvětlivek k přenášeným zprávám z obrázku 3.5.

JMÉNO – klient zasílá identifikační přezdívku.

HRY – klient zažádá o seznam existujících herních místností.

NOVÁ HRA – klient žádá o vytvoření nové hry.

JOIN – klient žádá o připojení do existující hry.

EXIT – klient žádá o odpojení ze hry.

MOVE – klient žádá o přesun agenta do jiného uzlu.

DROP – klient žádá o položení kreditů do aktuálního uzlu agenta.

TAKE – klient žádá o sebrání kreditů z aktuálního uzlu agenta.

ENQUIRY – klient oznamuje serveru, že v aktuálním tiku nechce vykonat již žádnou akci.

CHAT – klient posílá přes server zprávu jinému klientovi.

Tabulka 3.1: Popis síťového protokolu

Název požadavku	**Očekávaný stav**	**Požadavek**	**Odpověď**
JMÉNO	Identifikace	*newplayer;<jméno_hráče>*	*player;<jméno_hráče>;<barva _hráče>;none*
HRY	Lobby	*games*	*<seznam_her>*
NOVÁ HRA	Lobby	*getgame;<jméno_hry>;<jméno _hráče>;<parametry_hry...>*	*<stav_hry>*
JOIN	Lobby	*getgame;<jméno_hry>;<jméno _hráče>;<parametry_hry...>*	*<stav_hry>*
EXIT	Hra	*exit;<jméno_hráče>*	*exited*
MOVE	Hra	*move;<id_agenta>;<jméno _hráče>;<id_uzlu>*	*OK*
DROP	Hra	*drop;<id_agenta>;<jméno _hráče>;<počet_kreditů>*	*OK*
TAKE	Hra	*take;<id_agenta>;<jméno _hráče>;<počet_kreditů>*	*OK*
ENQUIRY	Hra	*enquiry;<id_agenta>;<jméno _hráče>;<počítadlo_tiku>*	*OK*
CHAT	Hra	*message;<typ_cíle>;<id _cíle>;<zpráva>*	*OK*

V rámci protokolu jsou přenášeny následující struktury:

- *jméno_hráče* – Řetězec reprezentující zvolené jméno hráče; nesmí přesahovat 20 znaků, obsahuje pouze alfanumerické znaky a mezery.
- *barva_hráče*, *barva_uzlu*, *barva_agenta* – Řetězec reprezentující hexadecimální vyjádření barvy v barevném modelu RGB; začíná vždy znakem # a má právě 7 znaků.
- *stav_hry* – Zpráva ve formátu JSON obsahující aktuální stav hry; blíže je její struktura popsána níže.
- *seznam_her* – Zpráva ve formátu JSON obsahující pole řetězců názvů her pod klíčem *games*.
- *jméno_hry* – Řetězec reprezentující jméno hry; nesmí přesahovat 20 znaků, obsahuje pouze alfanumerické znaky a mezery.
- *parametry_hry* – Seznam parametrů hry oddělený středníky v pořadí: podmínka výhry, vstup na cizí uzel, start v náhodném uzlu, limit agenta, generující uzly, viditelnost agentů.
- *id_agenta* – Unikátní číselný identifikátor agenta.
- *id_uzlu* – Unikátní číselný identifikátor uzlu.
- *typ_cíle* – Řetězcová informace o typu cíle; nabývá jedné z následujících hodnot: *single*, *all*, *group*.
- *id_cíle* – Identifikuje příjemce zprávy; představuje identifikátor agenta při typu cíle *single*, identifikátor skupiny při typu cíle *group* a jeho hodnota je ignorována při typu cíle *all*.
- *počet_akcí* – Počet zbývajících akcí, které má agent vykonat a požadavky na jejich vykonání již zaslal na server.
- *počet_kreditů* – Počet kreditů, které má agent vzít nebo položit.
- *info_agenta* – Zpráva ve formátu JSON obsahující informace o agentovi; blíže je její struktura popsána níže.
- *info_hráče* – Zpráva ve formátu JSON obsahující informace o hráči; blíže je její struktura popsána níže.
- *info_uzlu* – Zpráva ve formátu JSON obsahující informace o uzlu grafu; blíže je její struktura popsána níže.

- *info_hrany* – Zpráva ve formátu JSON obsahující informace o hraně grafu; blíže je její struktura popsána níže.
- *počítadlo_tiku* – Číselná hodnota udávající počet tiků uplynulých od začátku hry.
- *je_hráč_online* – Logická hodnota reprezentující přítomnost hráče ve hře; nabývá hodnoty 0 nebo 1.
- *souřadnice* – Číselná hodnota reprezentující souřadnici v rovině.
- *je_uzel_generující* – Logická hodnota reprezentující, zda je uzel generující; nabývá hodnoty 0 nebo 1.

Struktura stavu hry je popsána takto:

```
{
    "agents": {
        "nodes": {
            "0": [
                <info_agenta>
                ...
            ]
            "1": [
                <info_agenta>
                ...
            ]
            ...
        }
    }
    "players": [
        <info_hráče>
        ...
    ],
    "graph": {
        "nodes": [
            <info_uzlu>
            ...
        ],
        "edges": [
            <info_hrany>
            ...
```

```
        ]
    },
    "counter": <počítadlo_tiku>
}
```

Struktura informací o agentovi je popsána takto:

```
{
    "id": <id_agenta>,
    "carrying": <aktuální_počet_kreditů>,
    "carrylimit": <maximální_počet_kreditů>,
    "actions_count": <počet_akcí>,
    "player": <jméno_hráče>,
    "color": <barva_agenta>,
}
```

Struktura informací o hráči je popsána takto:

```
{
    "name" <jméno_hráče>,
    "color": <barva_hráče>,
    "ingame": <je_hráč_online>
}
```

Struktura informací o uzlu je popsána takto:

```
{
    "id": <id_uzlu>,
    "x": <souřadnice>,
    "y": <souřadnice>,
    "label": <popisek>,
    "color": <barva_uzlu>,
    "generating": <je_uzel_generující>,
    "owner": <jméno_hráče>,
    "homeof": <jméno_hráče>
}
```

Struktura informací o hraně je popsána takto:

```
{
    "from" <id_uzlu>,
    "to": <id_uzlu>
}
```

Na pravidelně zasílané aktualizace stavu hry klient odpovídat nemusí. Může však zaslat odpověď **ENQUIRY**, kterou oznámí, že již v aktuálním tiku nechce provádět žádnou akci. Server pro každou hru zná počet agentů a pokud je v aktuálním tiku přijata tato zpráva od všech, hra pokračuje následujícím tikem bez další časové prodlevy. Tím je dosaženo časové komprese, a tedy je umožněno hrát v režimu simulace. Pokud je alespoň jedním z účastníků hráč hrající prostřednictvím webového rozhraní, zprávu **ENQUIRY** neposílá.

Odpovědi uvedené v tabulce 3.1 se týkají pouze případů, kdy server provede požadovanou akci bez problémů. Pokud server není schopen akci vykonat (např. není v souladu s pravidly hry), odesílá odpověď *notOK*.

3.9.2 Protokol meziagentní komunikace

Za účelem umožnění vytvoření kteréhokoli multiagentního systému uvedeného v kapitole 2.2 bude navržen zapouzdřený protokol zajišťující komunikaci mezi jednotlivými agenty.

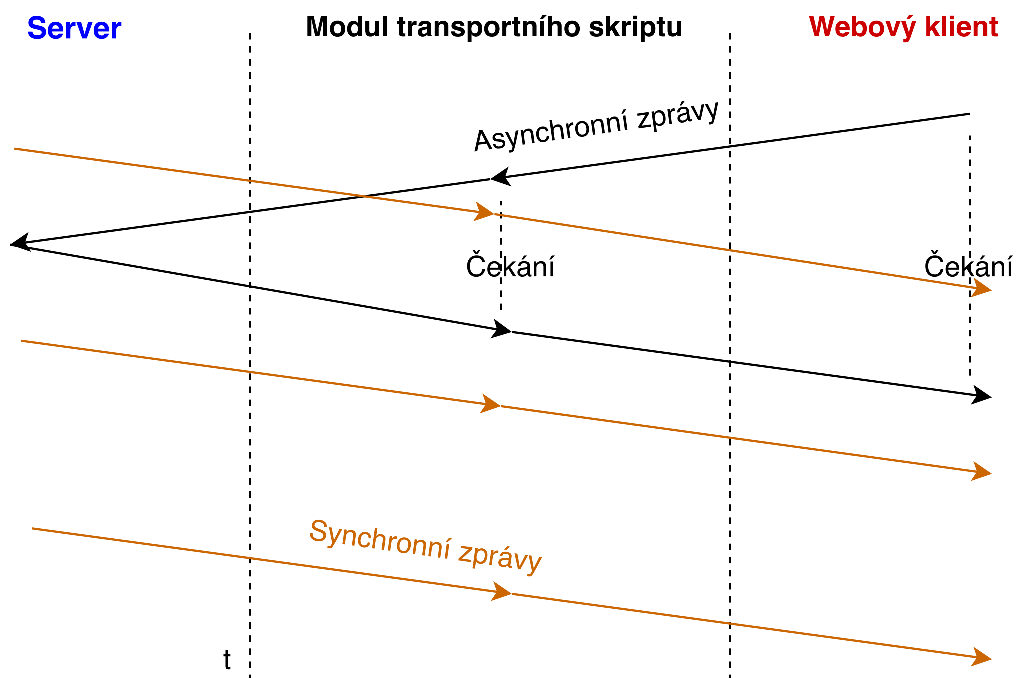
Adresace při komunikaci bude obdobou klasické síťové komunikace. Každý agent bude mít přiřazený svůj identifikátor, díky kterému bude možné zaslat individuální zprávu (*unicast*). Dále se může každý agent připojit do skupiny pro odběr skupinových zpráv (*multicast*). Všem agentům pak budou doručovány zprávy adresované všem v týmu (*broadcast*).

Pro meziagentní komunikaci byly navrženy tři základní druhy zpráv. Agenti si budou předávat informace o svém viditelném okolí ve formě podgrafu. Rovněž je obvykle nutné, aby si agenti vyměňovali informace o tzv. *záměru*, tedy vyšší abstrakci nad jednotlivými akcemi. Tím může být například „přivlastni uzel 5“, který obvykle sestává z několika elementárních akcí *move*, *take* a *drop*. Třetím druhem je výměna řetězcových zpráv, jejichž význam není předem určen a závisí na konkrétním uživateli, jaký význam přenášeným zprávám přiřadí.

3.10 Synchronizace v rámci komunikace

Vzhledem k požadavku na vizualizaci průběhu hry ve webovém rozhraní přibývá do komunikačního prostředí další prvek. Bude třeba zajistit synchronizaci dvou rozdílných datových toků. Jeden bude muset zajišťovat asynchronní požadavky, zatímco druhý bude synchronní s tiky hry. Modul transportního skriptu bude nutné vnitřně rozdělit podle těchto dvou požadavků. Po reakci na asynchronní požadavek bude modul muset zahájit čekání na příslušnou odpověď ze strany serveru a tuto odpověď následně předat modulu,

který je zdrojem požadavku. Synchronní kanál bude simplexní, vysílačem bude v tomto případě server a přijímačem každý z klientů. Současný průběh obou komunikací je zobrazen schématem 3.6.



Obrázek 3.6: Schéma komunikace; černou je znázorněna asynchronní zpráva; oranžové zprávy jsou synchronní.

4 Implementace

V kapitole *Implementace* budou popsány a odůvodněny postupy při vytváření aplikace a budou vysvětleny složitější nebo zajímavé části programu. Dokumentace je podrobnější, jelikož má aplikace sloužit k účelům výuky.

4.1 Vlákna

Program je rozčleněn do několika vláken, a to jak na straně serveru v C++, tak na straně klienta v jazyce Python.

Pro práci s vlákny v **C++** byla použita knihovna *std::thread*. Hlavní vlákno serveru v C++ má na starosti obsluhu komunikace s klienty. V tomto vlákně se nejdříve inicializuje *socket* [1] a zahájí se čekání na příchozí spojení. Poté je spuštěn nekonečný cyklus pro přijímání zpráv od klienta, jejich zpracování a podle obsahu zpráv volání příslušných ohlasových metod. Připojení více klientů je řešené pomocí funkce *select()*, která umožňuje čekání na více událostí současně.

Další vlákna na straně C++ jsou vytvářena dynamicky za běhu programu v závislosti na počtu vytvořených herních místností. Při každém vytvoření nové herní místnosti je spuštěno nové vlákno, jež má na starost obsluhu jedné konkrétní hry. Je tak zajištěn plynulý průběh všech her. Toto vlákno je reprezentováno metodou třídy *Game*. V tomto vlákně se zpracovává vše související se samotnou hrou, od přidávání a odebírání hráčů, zpracovávání jednotlivých tahů, přidávání nových agentů, až po generování zpráv pro klienty. Herní vlákno je ukončeno v případě, že skončí samotná hra, k čemuž může dojít za dvou okolností. V prvním případě, když jeden z hráčů porazí všechny ostatní, nastane logický konec hry a vlákno skončí. Ve druhém případě může nastat stav, kdy všichni hráči hru opustí, přestože hra stále běží. Pokud hru opustí poslední hráč, je ukončena jak hra, tak i vlákno, ve kterém hra běží.

V jazyce **Python** byla pro práci s vlákny použita knihovna *threading*. Jedno vlákno je ve třídě *Connector*, která obstaráva spojení se serverem v C++. Toto vlákno je vyčleněno pouze pro příjem zpráv, tedy volání funkce *recv()*. Vlákno přijme zprávu a podle jejího obsahu ji uloží do jedné z proměnných. Do konkrétní proměnné je zpráva přiřazena podle toho, zda se jedná o graf hry, jež přichází synchronně vždy s tikem hry, nebo v druhém případě o

odpověď na zprávu zaslanou klientem, jež může přicházet asynchronně. Samostatné vlákno pro příjem zpráv je nutné z toho důvodu, že z podstaty protokolu nelze předpokládat přesné uspořádání zpráv. To znamená, že na vícero požadavků nemusí odpovědi přijít v předpokládaném pořadí (např. kvůli konkurentnímu přístupu). Navíc může do tohoto toku zpráv zasahovat pravidelně odesílaný graf hry.

Předchozí vlákno je společné pro všechny implementace klientů, další vlákna jsou specifická pro agenty a hráče. Hráčem se v tomto případě rozumí uživatel hrající hru přes web, tedy využívající server Flask. Hráč v Pythonu potřebuje další dvě vlákna. Jedno slouží pro asynchronní komunikaci s JavaScriptem, kdy pomocí AJAX (Asynchronous JavaScript and XML) [10] zašle JavaScript požadavek Pythonu, ten ho vyřídí a pošle zpět odpověď. Druhé vlákno slouží pro pravidelné zasílání stavu hry opět JavaScriptu. To je zajištěno *websockety* [8] pomocí knihovny *Flask-SocketIO* [13].

Pokud se jedná o implementaci agenta, tomu stačí hlavní vlákno, ve kterém vykonává všechny své výpočty a akce. Pokud by měl však programátor důvod použít pro svého agenta vláken víc, nic mu v tom nebrání. Implementace agentů je zcela na uživateli systému.

4.2 Komponenty

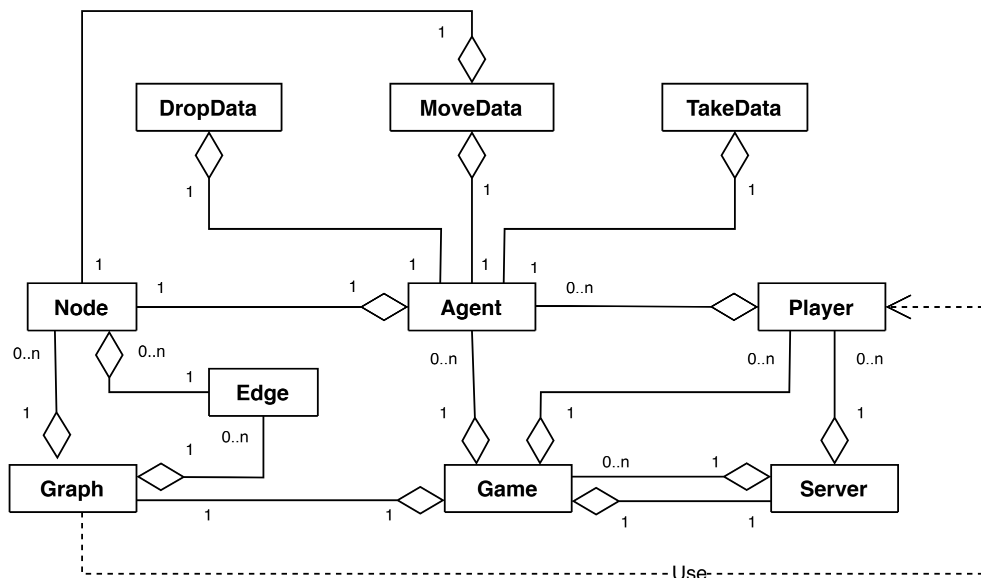
Zde budou popsány podstatné části programu a budou vysvětleny významné třídy a části programu s důležitou funkcí.

4.2.1 Server v C++

UML diagram tříd serveru v C++ je znázorněn na obrázku 4.1

Server v C++ se skládá z několika tříd. Mezi jedny z nejdůležitějších patří třída **Server**. Slouží ke spojení s klienty a prvotním reakcím na jejich požadavky. Třída vytvoří socket a začne naslouchat na IP adrese a portu, které jsou jí zadány jako parametry v konstruktoru. Do konstruktoru jsou po ošetření předávány ze vstupních parametrů programu. Třída zpracovává požadavky klientů a posílá jim příslušné odpovědi. Zároveň ověřuje validitu zpráv a pokud je na server zaslána zpráva neodpovídající protokolu, příslušný klientský socket odpojí. Podstatná je zde metoda *do_the_thing()*, která v nekonečném cyklu přijímá příchozí spojení a zprávy a na základě jejich obsahu volá další metody.

Další důležitou třídou je **Game**. Uchovává si veškeré informace o konkrétní



Obrázek 4.1: UML diagram tříd serveru v C++

hře. Její metoda *game()* je volána v samostatném vlákně. V této metodě se nachází cyklus, který je ukončen, pokud má zaniknout herní místnost. V každé iteraci zmíněného cyklu všichni agenti provedou svůj tah a všem je vygenerována a poslána příslušná zpráva obsahující stav uzlu, ve kterém se právě nachází. Zpráva rovněž obsahuje informaci o stavu sousedních uzlů. Ostatní metody třídy slouží k vykonávání tahů agentů, zajišťují veškerou herní logiku a shlukují informace do strukturovaných řetězců, z kterých je následně sestavena konečná zpráva o stavu hry.

Třída **Agent** si udržuje aktuální informace o stavu jednotlivých agentů a jejich sockety. Její metody vykonávají obsluhu tahů agenta.

Třída **Graph** reprezentuje graf právě jedné hry. Jedním z atributů je pole uzlů a druhým pole hran. Samotný graf pro konkrétní hru je generován v metodě *make_graph()* třídy Game. Třída Graph pak implementuje návrhový vzor přepravka [19], přičemž navíc obsahuje metodu pro vygenerování reprezentace grafu ve formátu JSON.

Stejně tak respektuje návrhový vzor přepravka třída **Player**. Její funkcí je uchování informací o jednotlivých týmech agentů. V případě hry na webu reprezentuje jednoho hráče a uchovává si jeho socket.

Další třídy **MoveData**, **DropData** a **TakeData** slouží k uchovávání informací o tazích agentů. Třída *Game* si uchovává pole instancí těchto tříd a využívá je k provedení tahů, které jsou v daný okamžik na řadě.

Instance třídy **Node** reprezentuje uzel grafu a obstarává změny počtu kreditů v něm a změnu jeho vlastníka. Instance třídy **Edge** představuje hranu v grafu a slouží pouze k akumulaci informací rovněž dle návrhového vzoru přepravka.

4.2.2 Python

V Pythonu je implementována třída **Connector**, která slouží k navázání spojení se serverem v C++. Za pomoci jejích metod jsou odesílány zprávy na server a to ať už se jedná o přihlášení nového hráče, vytvoření nové hry nebo o samotné požadavky na jednotlivé tahy agentů. Každý agent by si měl vytvořit instanci této třídy a komunikovat se serverem výhradně prostřednictvím jejích metod. Jelikož jsou příchozí zprávy ukládány do proměnných v separátním vlákně, byly pro synchronizaci uložení zprávy do proměnné s jejím čtením použity podmínkové proměnné.

Rozhraní pro komunikace mezi agenty je realizováno třídou **Communicator** nacházející se v souboru *communicator.py*. Při přenosu zpráv je využíváno instance třídy *Connector* k přenosu zpráv přes server v C++. Zprávy jsou posílány přes server v C++, protože se předpokládá spouštění agentů jednoho týmu z více zařízení. Server však slouží pouze jako prostředník a zprávy sám nijak neupravuje.

Pokud chce agent komunikační rozhraní používat, vytvoří si instanci třídy *Communicator* a zavolá její metodu *register_agent()*, jíž zašle jako parametr své identifikační číslo. Třída mu vytvoří vlastní „schránku“, kam mu ostatní agenti budou posílat zprávy a on si je odtud bude vybírat. Schránky agentů jsou implementovány jako asociativní pole, jehož klíče jsou identifikační čísla jednotlivých agentů a hodnotami jsou pole, jež obsahují jednotlivé zprávy pro konkrétního agenta.

Agent se také může zaregistrovat do skupiny voláním funkce *register_to_group()*, která jako parametry přebírá identifikační číslo registrujícího se agenta a identifikační číslo skupiny, do které se chce přidat.

Dále jsou implementovány metody, které umožňují agentům posílat graf hry, záměr, nebo vlastní znakový řetězec, jehož formát si programátor může

nadefinovat sám. Každá metoda pro zaslání jednoho druhu zprávy je pak implementována třikrát, a to pro posílání jednomu agentovi, skupině agentů, nebo všem agentům. Třída dále obsahuje metodu *get_waiting_message()*, která agentovi umožňuje vybrat si právě jednu zprávu ze schránky. Po přečtení zprávy agentem se tato zpráva odstraní z agentovy schránky. Pokud se již agent nechce vyskytovat v některé ze skupin, může použít metodu *leave_group()*.

V souboru **routes.py** se nachází mechanismus pro komunikaci s jazykem JavaScript. Jednotlivé funkce jsou volány pomocí AJAX. Uvnitř těchto funkcí se zpravidla volají metody třídy *Connector*. Následně je JavaScriptu zasílána odpověď. To neplatí pro jedinou funkci *test_connect()*, která je volána při připojení websocketu a odesílá JavaScriptu odpověď, že připojení proběhlo v pořádku.

Soubor **__init__.py** slouží rovněž pro případ spouštění hry na webu. Provádí se v něm ošetření vstupních parametrů programu. Následně jsou v něm inicializovány instance tříd *Flask* a *SocketIO* pro navázání komunikace s JavaScriptem.

Soubor **main.py** má na starost spuštění všech vláken potřebných pro komunikaci s webovým rozhraním.

4.3 Transparentnost přenášených zpráv

Při přenosu zpáv mezi serverem v C++ a klientem v jazyce Python a nazpátek bylo třeba zajistit, že všechny zprávy přijdou v celku. To je řešeno v metodě *recv_msg()* třídy *Connector* pro přenos z serveru v C++ ke klientovi a v metodě *do_the_thing()* třídy *Server* pro přenos v opačném směru následujícím způsobem. Na straně, která zprávu posílá, je zpráva vymezena zleva i zprava složenými závorkami. Na straně příjemce zprávy by pak bylo možné rozpoznat konec zprávy podle ukončovací závorky. Některé zprávy jsou však ve tvaru JSON, proto se závorky nacházejí i uvnitř zprávy. To bylo vyřešeno počítáním levých a pravých závorek při příjmu zprávy. Příjem probíhá po jednom bytu. Byty jsou za sebou řazeny do řetězce, jenž představuje výslednou zprávu. Když je napočítán stejný počet příchozích levých a pravých složených závorek, zpráva se považuje za kompletní. První a poslední závorka se nepřidávají do zprávy, protože jsou uměle přidávány před jejím odesláním. Ve jméně hráče (týmu) nebo názvu hry jsou závorky

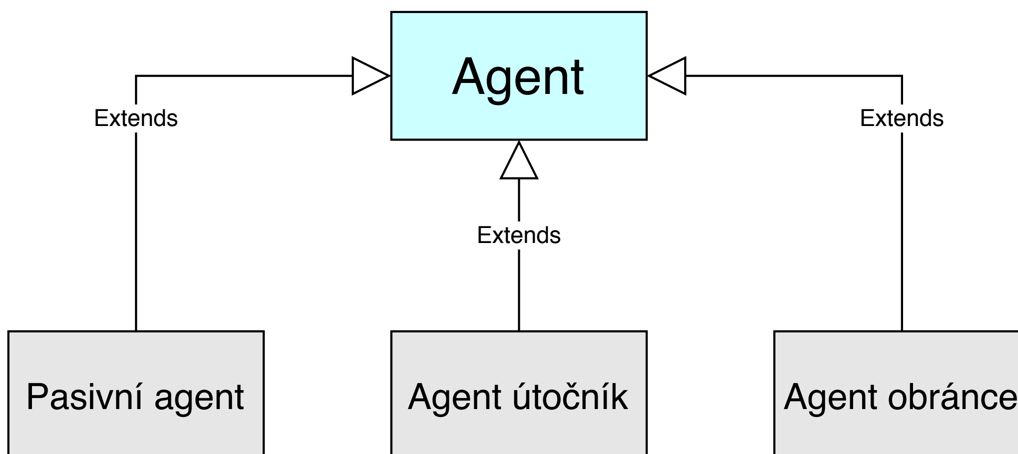
zakázány a jsou z řetězců mazány ještě před vytvořením zprávy pro odeslání na server. Pokud přijatá zpráva nezačíná závorkou, klient se po přijetí odpojí od serveru nebo při přijetí takovéto zprávy serverem server ukončí spojení s příslušným klientem.

4.4 Implementovaní agenti

Pro otestování funkčnosti systému bylo implementováno několik rozdílných typů agentů. Tito agenti budou popsáni v následující části. Pochopení jejich implementace je důležité pro následné porozumění výsledkům experimentů.

4.4.1 Struktura agentů

Pro usnadnění vytváření agentů byla v jazyce Python vytvořena třída *Agent*. Všichni agenti od této třídy dědí a přepisují metodu *run()*. Třída *Agent* obsahuje často používané konstrukce napříč různými agenty. Metoda *set_game_parameters()* nastaví parametry pro vytvářenou hru a metoda *initialize()* vytvoří spojení se serverem v C++ a přihlásí agenta k týmové komunikaci. Pro vytvoření a spuštění samostatného vlákna pro běh agenta je implementována metoda *start()*, metoda *wait_for_end()* pak čeká na skončení tohoto vlákna. Dědičnost je zobrazena na obrázku 4.2.



Obrázek 4.2: UML diagram dědičnosti jednotlivých agentů od třídy *Agent*

Třída *Team* reprezentuje jeden tým agentů. Slouží k usnadnění spouštění agentů stejného týmu z jednoho místa. Agenty lze spustit i bez použití této

třídy. Třída *Team* byla použita při vytváření experimentů. Spuštění a řádné ukončení běhu agentů zajišťuje metoda *play()*. Metoda *get_game_outcome()* vrací řetězec informující o výsledku hry pro daný tým a metoda *get_game_counter()* vrací počet herních tiků uplynulých od začátku do konce hry. V každém z jednotlivých experimentů jsou jednotlivé týmy spouštěny v samostatných vláknech.

4.4.2 Pasivní agent

Skript pasivního agenta si nejdříve vytvoří vlastní instanci třídy *Connector* pro komunikaci se serverem. Následně vytvoří tým nebo se připojí do již existujícího. Dále vytvoří novou hru či se připojí do již vytvořené. Agent dále vyčkává na konec hry, aniž by cokoli dělal. Nenavazuje komunikaci s jinými agenty a při konci hry pouze vypíše, zda jeho tým vyhrál nebo prohrál. Ze skriptu pasivního agenta vycházejí všichni ostatní implementovaní agenti a jeho úkolem je poskytnout příklad základní struktury agenta.

4.4.3 Agent průzkumník

Agent průzkumník má za úkol procházet graf a informace o svém okolí posílat všem agentům v týmu. Pro implementaci agenta průzkumníka byla použita nerekurzivní verze algoritmu *prohledávání do hloubky* [16]. Do zásobníku je ukládána cesta v podobě identifikačních čísel uzlů. Tento algoritmus byl zvolen, jelikož je zamýšleno použít agenta zejména pro nalezení domovského uzlu nepřátelského týmu, u něhož se nepředpokládá, že bude v blízké vzdálenosti od domovského uzlu týmu, v němž se agent nachází.

Při každém tiku hry je aktualizován stav okolí uzlu, ve kterém se agent nachází voláním metody *get_game_msg()*. Tento stav je následně zaslán všem agentům v týmu, kteří jsou přhlášení k používání komunikace. Odesílání stavu grafu se děje za pomoci rozhraní *Communicator* voláním metody *sent_graph_to_all()*. Pro kooperaci s agentem útočníkem (viz část 4.4.4) se navíc při objevení prvního generujícího uzlu nebo prvního domovského uzlu protivníka zašle všem agentům zpráva obsahující popis cesty z domovského uzlu do domovského uzlu protivníka.

4.4.4 Agent útočník

Agent útočník po přihlášení do hry čeká na zprávu od průzkumníka o nalezeném volném generujícím uzlu za využití aktivního čekání, opakovaným voláním metody *get_waiting_message()* třídy *Communicator*. Po přijetí zprávy

sebere v domovském uzlu kredity a přesune se do generujícího uzlu. Ten si zabere položením jednoho kreditu. Dále čeká na zprávu o nalezeném domovském uzlu protivníka. Když tuto zprávu obdrží, střídavě do protivníkova uzlu nosí kredity ze zabraného generujícího a svého domovského uzlu, dokud protivníkův uzel nezabere.

Tento agent je použitelný pouze při spuštění s agentem průzkumníkem ve stejném týmu. Sám graf neprohledává a bez agenta průzkumníka je jeho chování totožné s pasivním agentem.

4.4.5 Agent útočník–podvodník

Tento agent byl vytvořen čistě pro účely testování a způsob, jakým je vytvářen, bude v ostře nasazené verzi systému zakázán. Agent je vytvořen s parametrem „web-server“ nastaveným na hodnotu 1. Tím serveru říká, že se jedná o webový server. Z důvodu vizualizace v rozpracované podobě jsou webovému serveru zasílány celé grafy her. To znamená, že viditelnost agenta je maximální možná.

Agent útočník-podvodník se chová stejně jako agent útočník (viz část 4.4.4), ale protože vidí celý graf, nepotřebuje k sobě agenta průzkumníka, který by pro něj potřebné uzly hledal. Agent si v grafu hry najde volný generující uzel a domovský uzel protivníka a bez prodlevy začne jednat. Tento agent byl uměle vytvořen kvůli porovnání s agentem útočníkem, který je závislý na chování agenta průzkumníka a tím ověření vlivu parametru „viditelnost“ na průběh hry.

4.4.6 Agent obránce

Pro agenta obránce je důležitá, jak již z jeho názvu vyplývá, obrana domovského uzlu. Agent implementuje algoritmus *prohledávání do šířky* [16] s počátkem v domovském uzlu. Do zásobníku se ukládají celé cesty z domovského uzlu a mezi uzly se pak přechází vždy přes tento domovský uzel, v němž agent sebere kredity. Takto se prochází všechny uzly v grafu. Pokud je uzel, ve kterém se zrovna nachází, volný, položí do něj kredity. V případě spuštění hry se zapnutým parametrem „vstup na cizí uzel“ (viz sekce 3.3.2) si tak domovský uzel v podstatě obrní a protivníkovi tak zkomplikuje cestu k tomuto uzlu. Pokud je tento parametr vypnutý, může agent sloužit k zabírání uzlů, což je prospěšné pro všechna nastavení parametrů hry.

4.5 Vizualizace

Vizualizace byla vytvořena zejména k průběžnému ověřování funkčnosti systému. Celá vizualizace sestává ze dvou částí. První z nich je webový server v jazyce Python, který slouží jako prostředník při přenášení zpráv z webu na server v C++ a zpět.

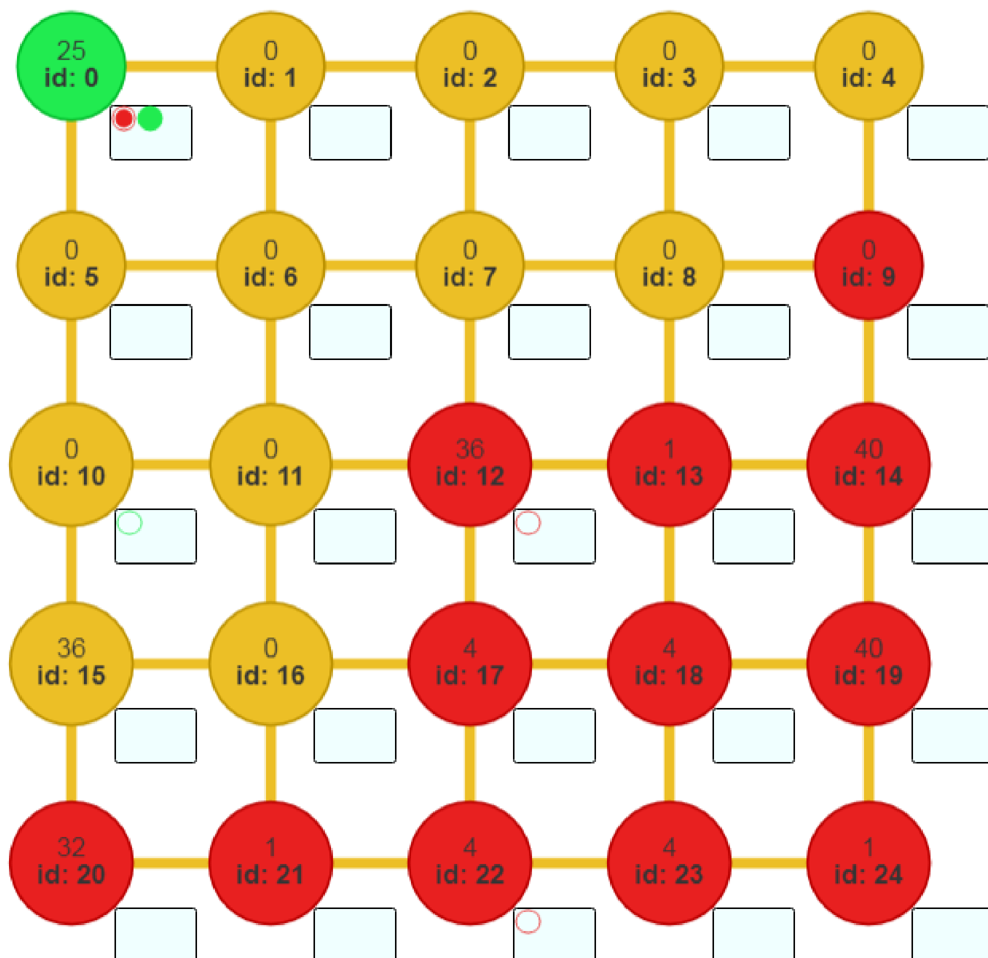
Spuštění webového serveru se děje v souboru *main.py*. V tomto souboru jsou nejdříve vytvořena dvě vlákna, a to pro synchronní a asynchronní komunikaci mezi serverem v C++ a webem. Veškeré funkce pro webovou vizualizaci se nachází v souboru *main.js*. Mezi tyto funkce patří zejména funkce *drawGraph()* pro prvotní vykreslení grafu a *updateGraph()* pro aktualizaci grafu. V nich se pracuje s knihovnou *vis.js* [18], která umožňuje jak vykreslení libovolného grafu ze zadaného řetězce ve formátu JSON, tak jeho aktualizaci.

Pro asynchronní dotazy, mezi něž může patřit například přihlášení hráče, vytvoření hry, nebo zasílání tahů, je použit AJAX (Asynchronous JavaScript and XML) [10]. JavaScript své asynchronní dotazy předává přes *routes.py* třídě *Connector*, která se postará o jejich zaslání na server v C++. Odpověď na dotaz se stejnou cestou v opačném směru dostane zpět na web. Pro synchronní zasílání zpráv, jímž je zpravida posílání aktualizovaného stavu grafu, bylo využito knihovny *Flask-SocketIO*. Její výhodou je možnost vytvořit websocket pro pouze jednosměrnou komunikaci, narozdíl od AJAX, který vždy čeká na odpověď. Tím bylo umožněno synchronní zasílání zpráv ze serveru v C++ na web bez předchozího požadavku z webu.

Na webu je možné přihlásit se pod jménem „observer“, což umožní uživateli připojit se do hry a sledovat ji, aniž by se jí sám účastnil. Samotná webová vizualizace se skládá z několika obrazovek. Všechny obrazovky a návod k používání jsou popsány v uživatelské příručce. Všechny události jsou obstarávány knihovnou *jQuery* [6]. Protože vizualizace nebyla součástí zadání a znalost její podrobné implementace není podstatná pro používání systému, nebude popisována do větších podrobností.

5 Dosažené výsledky

K otestování funkčnosti systému bylo vytvořeno několik experimentů, ve kterých se proti sobě utkávají týmy různých pravidlově řízených agentů v různě parametrizovaných hrách. Cílem bylo ověřit jak fungování celého systému a simulace, tak vliv různých sestav týmů a změn parametrů na výherní strategie. Veškeré experimenty byly spouštěny ve verzi 3.7.4 jazyka Python. Pro každý experiment bylo na počátku hry uměle přidáno do domovských uzlů obou týmů třicet kreditů z důvodu umožnění agentům využít jejich taktik, aniž by hra skončila po prvním přenesení kreditů do protivníkova uzlu. Pro ukázku je výstřižek ze hry „experiment5“ z vizualizace v módu „observer“ k vidění na obrázku 5.1.



Obrázek 5.1: Náhled na hru „experiment 5“ z pozice „observer“

V každém experimentu byl sledovaným měřítkem počet výher každého z týmů při opakovaném spouštění experimentu a počet herních tiků, za který byla celá hra odehrána. Každý experiment byl spuštěn právě stokrát a při každém spuštění byly zaznamenány naměřené hodnoty. Ty jsou k nalezení v tabulce 5.1. Remízou se rozumí stav, kdy si agenti přebrali uzly ve stejný čas navzájem, nebo nebyl vlivem počátečních podmínek žádný z agentů schopen vyhrát hru do uběhnutí tisíce tiků. Počet tiků je uveden ve formátu: *<průměrný počet tiků z 100 her>* ± *<směrodatná odchylka>*.

Experiment 1

- Nastavení parametrů hry:
  - Podmínka výhry – Take home.
  - Vstup na cizí uzel – Bez penalizace.
  - Start v náhodném uzlu – Ano.
  - Limit agenta – 10.
  - Generující uzly – jednou za 5 tiků.
  - Viditelnost – 1.

V tomto experimentu se proti sobě utkal pasivní agent s agentem útočníkem-podvodníkem. Nastavení parametrů je kromě podmínky výhry výchozí. Podmínka hry byla takto zvolena, protože cílem agenta útočníka–podvodníka je pouze zabrat domovský uzel protivníka. Očekávaným výsledkem experimentu bylo stoprocentní vítězství agenta útočníka–podvodníka. Z tabulky 5.1 lze vypozorovat naplnění tohoto očekávání, čímž byla ověřena základní funkčnost systému.

Experiment 2

- Nastavení parametrů hry:
  - Podmínka výhry – Take home.
  - Vstup na cizí uzel – Bez penalizace.
  - Start v náhodném uzlu – Ano.
  - Limit agenta – 10.
  - Generující uzly – jednou za 5 tiků.
  - Viditelnost – 1.

Tento experiment, ve kterém se proti sobě utkal pasivní agent s agentem útočníkem, měl rovněž za úkol ověřit funkčnost systému. To se znovu potvrdilo téměř stoprocentním vítězstvím agenta útočníka. Výskyt remíz byl způsoben tím, že cesta do domovského uzlu protivníka, nelezená agentem průzkumníkem, byla tak dlouhá, že agent útočník nestíhal nosit do protivníkova uzlu větší počet kreditů, než se v něm za dobu putování agenta útočníka vygeneroval. Dále měl tento experiment v porovnání s experimentem 1 ukázat vliv omezené viditelnosti agenta na rychlost hry. Experiment ukázal očekávané, a sice že počet tiků na odehrání hry byl u agenta útočníka-podvodníka řádově nižší.

Experiment 3

- Nastavení parametrů hry:
  - Podmínka výhry – Take home.
  - Vstup na cizí uzel – Bez penalizace.
  - Start v náhodném uzlu – Ano.
  - Limit agenta – 10.
  - Generující uzly – jednou za 5 tiků.
  - Viditelnost – 1.

Souboj dvou stejných týmů agentů měl poukázat na závislost výsledku hry na startu v náhodném uzlu a náhodném rozmístění generujících uzlů. Výsledky sto pokusů jsou téměř vyvážené. Při počtu pokusů blížícího se k nekonečnu by se měly statistiky výher obou týmů limitně blížit ke stejnému procentu (zbytek by připadl na remízy). Nízký počet remíz je způsoben

chováním agenta průzkumníka v kombinaci s náhodným rozmístěním generujících uzlů (je malá pravděpodobnost, že agenti průzkumníci obou týmů naleznou stejně dlouhé cesty do generujícího i do protivníkova domovského uzlu).

Experiment 4

- Nastavení parametrů hry:
  - Podmínka výhry – Take home.
  - Vstup na cizí uzel – Bez penalizace.
  - Start v náhodném uzlu – Ano.
  - Limit agenta – 10.
  - Generující uzly – jednou za 5 tiků.
  - Viditelnost – 1.

Cílem experimentu bylo opět zjistit závislost výsledků na startu v náhodném uzlu a náhodném rozmístění generujících uzlů. Tentokrát se však oproti experimentu 3 vyskytl větší počet remíz. Stalo se tak, když našli agenti své generující uzly ve stejné vzdálenosti od domovského uzlu protivníka. Průměrný počet tiků na hru byl stále nižší než u předchozího experimentu agentů s nižší viditelností (viz tabulka 5.1), přestože rozdíl už nebyl tak vysoký jako u her s pasivními agenty.

Experiment 5

- Nastavení parametrů hry:
  - Podmínka výhry – Take home.
  - Vstup na cizí uzel – S penalizací.
  - Start v náhodném uzlu – Ne.
  - Limit agenta – 4.
  - Generující uzly – jednou za 20 tiků.
  - Viditelnost – 1.

V tomto experimentu bylo cílem zapojit do týmu agenta obránce. Pro jeho užití bylo zvoleno zapnutí parametru „vstup na cizí uzel“, aby bylo

možné skutečně využít agentovu obrannou strategii. Ukázalo se, že při zvyšování hodnoty parametru „generující uzly“ a snižování hodnoty parametru „limit agenta“ se počet výher přikláněl více na stranu týmu s obranným agentem. Nakonec byla ponechána konfigurace parametrů, ve které je patrný vliv prezence agenta obránce na výsledky her.

Experiment 6

- Nastavení parametrů hry:
  - Podmínka výhry – Take home.
  - Vstup na cizí uzel – S penalizací.
  - Start v náhodném uzlu – Ne.
  - Limit agenta – 15.
  - Generující uzly – jednou za 5 tiků.
  - Viditelnost – 1.

Při konfiguraci parametrů hry s vyšší hodnotou parametru „limit agenta“ a nižší hodnotou parametru „generující uzly“ oproti experimentu 5 byl tým s agentem obráncem ve většině případů poražen. V porovnání s experimentem 5 se úspěšnost celého týmu s agentem obráncem značně snížila. Z toho lze usoudit, že úspěch sestaveného týmu je závislý na parametrech „limit agenta“ a „generující uzly“.

Experiment 7

- Nastavení parametrů hry:
  - Podmínka výhry – Take home.
  - Vstup na cizí uzel – Bez penalizace.
  - Start v náhodném uzlu – Ano.
  - Limit agenta – 10.
  - Generující uzly – jednou za 5 tiků.
  - Viditelnost – 1.

Experiment měl za úkol zjistit, zda bude tým se dvěma agenty útočníky ve výhodě oproti týmu s pouze jedním útočníkem. Z výsledků je patrné, že hlavní roli opět hrál parametr „start v náhodném uzlu“ a náhodné rozmístění generujících uzlů.

Experiment 8

- Nastavení parametrů hry:
  - Podmínka výhry – Take home.
  - Vstup na cizí uzel – Bez penalizace.
  - Start v náhodném uzlu – Ano.
  - Limit agenta – 5.
  - Generující uzly – jednou za 10 tiků.
  - Viditelnost – 2.

Při změně některých parametrů oproti experimentu 7 se stejnými týmy agentů byly pozorovány stejně vyvážené výsledky. Z obou experimentů by se tedy dalo usoudit, že větší počet agentů útočníků nemá vliv na výsledek hry. Z výsledků se dá rovněž pozorovat, že změna parametrů mohla mít vliv na kvantitu výskytu remíz.

Rychlost simulace

Experimenty byly spouštěny v režimu simulace. Nejrychleji proběhlo sto spuštění experimentu 1 a to za 144 sekund (1.4s na jedno spuštění). Z toho vychází přibližně rychlost 28 tiků hry za sekundu. Nejdéle potom trvala spuštění experimentu 5, jehož 100 pokusů proběhlo za 3 257 sekund (cca 33s na jedno spuštění). Z těchto hodnot lze vypočítat přibližnou rychlost 10 tiků hry za sekundu.

Experimenty byly měřeny na Lenovo ThinkPad E570, 8GB RAM DDR4, Intel Core i5-7200U (2.5GHz).

Tabulka 5.1: Výsledky jednotlivých experimentů; hodnoty zaokrouhleny na celá čísla.

#	**Složení týmu 1**	**Složení týmu 2**	**Výhry týmu 1**	**Výhry týmu 2**	**Remízy**	**Počet tiků na hru**
1	Pasivní	Útočník-podvodník	0%	100%	0%	39 ± 14
2	Pasivní	Útočník Průzkumník	0%	96%	4%	231± 215
3	Útočník Průzkumník	Útočník Průzkumník	42%	57%	1%	65 ± 30
4	Útočník-podvodník	Útočník-podvodník	28%	45%	27%	28 ± 6
5	Útočník Průzkumník	Útočník Průzkumník Obránce	27%	64%	9%	321 ± 263
6	Útočník Průzkumník	Útočník Průzkumník Obránce	97%	3%	0%	40 ± 14
7	Útočník Průzkumník	2x Útočník Průzkumník	49%	43%	8%	44 ± 19
8	Útočník Průzkumník	2x Útočník Průzkumník	46%	54%	0%	96 ± 65

6 Závěr

Byl vyvinut multiagentní systém umožňující implementaci agentů využívajících umělou inteligenci. Rovněž byla umožněna simulace her a systém splňuje i ostatní požadavky z kapitoly 3.1.

Prostřednictvím experimentů byla ověřena funkčnost systému, včetně meziagentní komunikace. Zasílání zpráv mezi agenty bylo otestováno mezi agentem průzkumníkem a agentem útočníkem. Taktéž byl otestován vliv všech parametrů hry na výsledek hry. Z výsledků experimentů 5 a 6 je patrné, že různá nastavení parametrů hry převrací výsledky her ve prospěch jednoho, či druhého týmu pravidlově řízených agentů. Neexistuje tedy tým pravidlově řízených agentů s univerzální výherní strategií a je nutné použití umělé inteligence pro přizpůsobování týmů agentů herním podmínkám.

Navíc byla implementována webová vizualizace umožňující hráčům zahrát si buď proti sobě nebo proti naimplementovanému týmu agentů. Součástí vizualizace je i mód pro sledování hry bez přímé účasti. To usnadňuje uživateli pochopení výsledků her mezi týmy agentů.

Systém byl implementován co nejobecněji, aby byl rozšiřitelný pro různé typy úloh. Systém byl testován na neorientovaném grafu uspořádaném do mřížky a to zejména z důvodu možnosti propojení s vizualizací. Do budoucna je v plánu modifikovat graf na náhodně generovaný souvislý graf, čímž přibyde do hry další role náhody. Předpokládá se, že je systém implementován dostatečně obecně a tato změna nebude vyžadovat zásadní zásah do již existující infrastruktury.

Dalších možných rozšíření systému by se dalo vymyslet hodně. Například umožnění agentům předávat si mezi sebou kredity bez položení do uzlu nebo dát jednotlivým agentům možnost vidět dál za cenu snížení jejich limitu a spousta dalších.

Systém je v plánu dále rozvíjet a v budoucnu i zapojit do praktické části výuky předmětu KIV/ISW na Západočeské univerzitě v Plzni.

Literatura

[1] IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7. *IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)*. 2018, s. 1–3951.

[2] *AI - Agents & Environments* [online]. TutorialsPoint, 2016. [cit. 4.3.2020]. Dostupné z: `https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_agents_and_environments.htm`.

[3] Banks, J. – Carson, J. S. – Barry, L. Discrete-event system simulation (fourth edition), 2005.

[4] Bray, T. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 8259, RFC Editor, prosinec 2017. Dostupné z: `https://tools.ietf.org/html/rfc8259`.

[5] Bray, T. et al. Extensible Markup Language (XML) 1.0 (Third Edition). fifth edition of a recommendation, W3C, 2008. https://www.w3.org/TR/2008/REC-xml-20081126/.

[6] Chaffer, J. – Swedberg, K. *JQuery Reference Guide: A Comprehensive Exploration of the Popular JavaScript Library*. Packt Publishing, 2007. ISBN 978-1-84719-381-0.

[7] Davidson, E. M. et al. Applying multi-agent system technology in practice: Automated management and analysis of SCADA and digital fault recorder data. *IEEE Transactions on Power Systems*. 2006, 21, 2, s. 559–567.

[8] Fette, I. et al. The WebSocket Protocol. RFC 6455, RFC Editor, prosinec 2011. Dostupné z: `https://tools.ietf.org/html/rfc6455`.

[9] Finin, T. et al. KQML as an agent communication language. In *Proceedings of the third international conference on Information and knowledge management*, s. 456–463, 1994.

[10] Garrett, J. J. Ajax: A new approach to web applications. *Adaptive Path*. 2005.

[11] Gemrot, J. et al. Pogamut 3 can assist developers in building AI (not only) for their videogame agents. In *International Workshop on Agents for Games and Simulations*, s. 1–15. Springer, 2009.

[12] *GNU General Public License, version 3* [online]. Free Software Foundation, Inc, 2007. [cit. 17.4.2020]. Dostupné z: https://www.gnu.org/licenses/gpl-3.0.html.

[13] Grinberg, M. *Flask-SocketIO* [online]. 2018. [cit. 24.11.2019]. Dostupné z: https://flask-socketio.readthedocs.io/en/latest.

[14] ISO/IEC. ISO/IEC 7498-1: 1994 information technology–open systems interconnection–basic reference model: The basic model. *International Standard*. 1996.

[15] Juneja, D. – Jagga, A. – Singh, A. A review of FIPA standardized agent communication language and interaction protocols. *Journal of Network Communications and Emerging Technologies*. 2015, 5, 2, s. 179–191.

[16] Kozen, D. *The Design and Analysis of Algorithms – Depth-First and Breadth-First Search*. Springer, New York, NY, 1992. ISBN 978-1-4612-4400-4.

[17] Kubík, A. Inteligentní agenty (Intelligent Agents), 2004.

[18] Mulder, A. – Jong, J. *vis.js – A dynamic, browser based visualisation library* [online]. Almende B.V., 2016. [cit. 10.12.2019]. Dostupné z: https://visjs.org.

[19] Pecinovskỳ, R. *Návrhové vzory*. Albatros Media as, 2015. ISBN 978-80-251-1582-4.

[20] Postel, J. Transmission Control Protocol. RFC 793, RFC Editor, září 1981. Dostupné z: https://tools.ietf.org/html/rfc793.

[21] Sycara, K. P. Multiagent Systems. *AI Magazine*. Jun. 1998, 19, 2, s. 79. doi: 10.1609/aimag.v19i2.1370. Dostupné z: https://wvvw.aaai.org/ojs/index.php/aimagazine/article/view/1370.

A Uživatelská příručka

Softwarové požadavky

Pro překlad serveru je potřeba mít nainstalovaný *CMake* verze 3.0 a vyšší a kompilátor s podporou *C++11*. Pro spuštení klienta je třeba mít nainstalovaný *Python 3.7* a vyšší a správce balíčků *pip*.

Překlad

Pro sestavení serveru zadejte v kořenové složce serveru *serverbaka* příkazy:

```
cmake .
cmake --build .
```

Výsledkem překladu bude spustitelný soubor *serverbaka.exe*.

K instalaci potřebných knihoven pro klientskou část v jazyce Python je připraven soubor *requirements.txt*. Po zadání následujícího příkazu budou všechny potřebné knihovny nainstalovány:

```
pip install -r requirements.txt
```

Spuštění

Server bude spuštěn pomocí vytvořeného souboru *serverbaka.exe*. Může být spuštěn s parametrem udávajícím číslo portu na kterém bude naslouchat. Pokud nebude parametr zadán, bude server naslouchat na portu 7890. Příklad spuštění serveru:

```
./serverbaka.exe 9901
```

Pro spuštění programu v Pythonu představujícího webový server pro umožnění vizualizace je třeba zadat ve složce *flask_server* příkaz se dvěma parametry:

```
python main.py <server IP> <server port>
```

Příklad takového spuštění:

```
python main.py 192.168.0.2 7890
```

Pak je možné ve webovém prohlížeči spustit vizualizaci zadáním URL adresy *http://127.0.0.1:8080*.

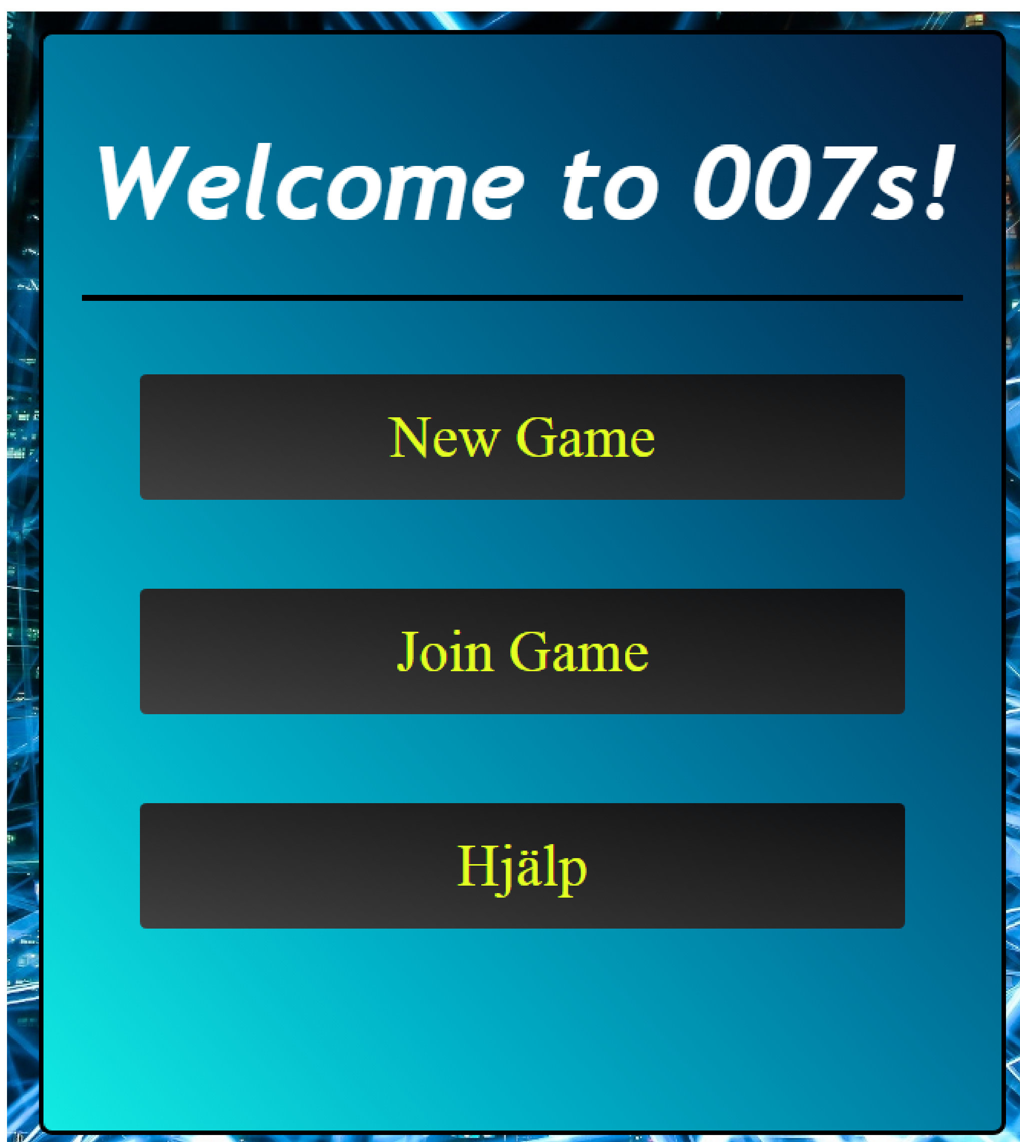
Pokud chce uživatel spustit některý z experimentů, popsaných v kapitole 5, je třeba zadat rovněž ve složce *flask_server* následující příkaz, kde ID značí číslo experimentu:

```
python experiment_<ID>.py
```

Webové rozhraní

Při prvotním načtení stránky se objeví obrazovka s přihlášením. Ta od uživatele žádá zadání jeho přezdívky, která by neměla přesahovat dvacet znaků a obsahovat jiné, než alfanumerické znaky. Pokud chce uživatel pouze na hry nahlížet, ale sám se jich neúčastnit, použije jako přezdívku jméno „observer“. Následné přihlášení je provedeno po kliknutí na tlačítko „Log in“.

Po přihlášení se uživatel dostane do menu. Zde může vytvořit novou hru kliknutím na tlačítko „New Game“ nebo si nechat vypsat již existující hry kliknutím na tlačítko „Join Game“. Pokud potřebuje zjistit, jak se hra hraje, může si nechat vypsat pravidla hry kliknutím na tlačítko „Hjälp“.

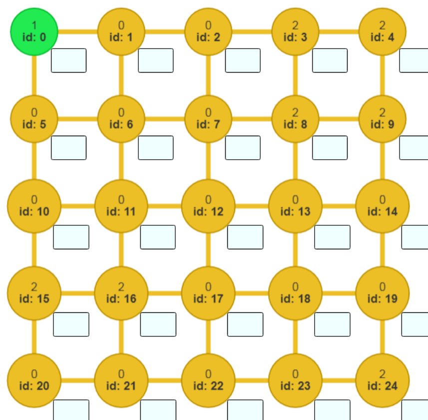


Pokud klikne uživatel na „New Game“, zobrazí se mu formulář pro nastavení parametrů hry. Zde je možné nastavit každý z parametrů popsaných v 3.3.2, kromě viditelnosti, ta je při vizualizaci nastavena na maximální možnou hodnotu.

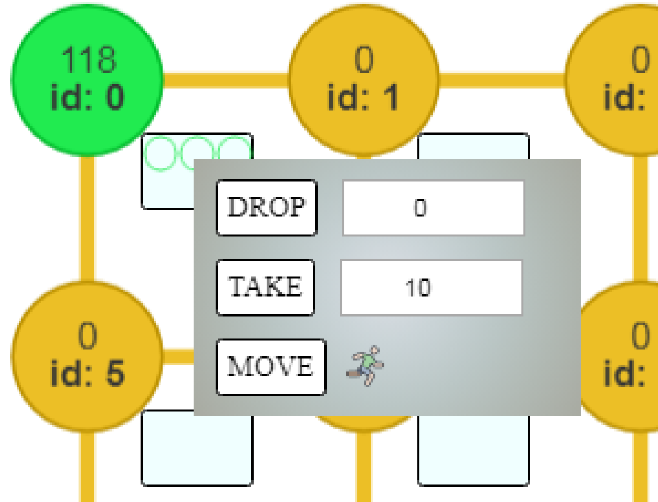
New Game

Name of game: Tournament
Winning mode: Take all
Enemy node dec:
Start in random node:
Agent carry limit: 10
Ticks per generate: 5
Create

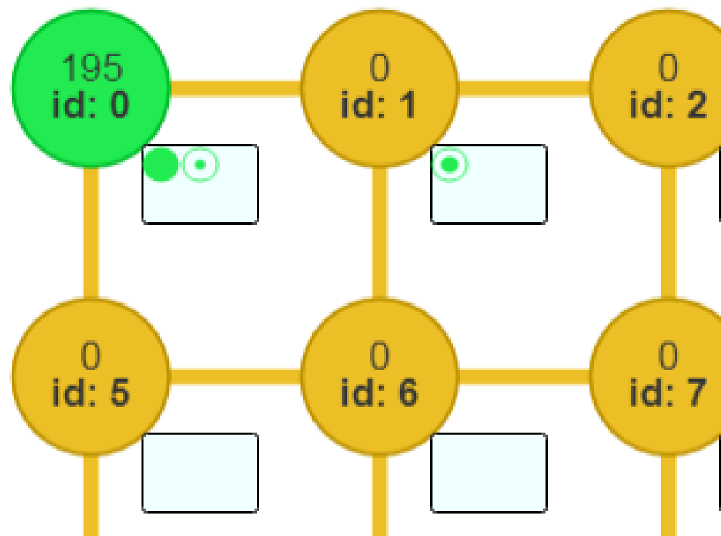
Po vyplnění formuláře a kliknutí na tlačítko „Create“ je založena hra a uživateli je zobrazen graf hry, dále v levém horním rohu seznam aktuálně připojených hráčů do této hry a v pravém horním rohu tlačítko pro odchod ze hry. Na pravo od grafu se nachází tlačítko „Add agent“ pro přidání nového agenta do domovského uzlu. Domovský uzel hráče má barvu shodnou s barvou uvedenou v seznamu hráčů. Každý uzel je označen identifikačním číslem a nad ním se nachází číslo udávající aktuální počet kreditů v uzlu.



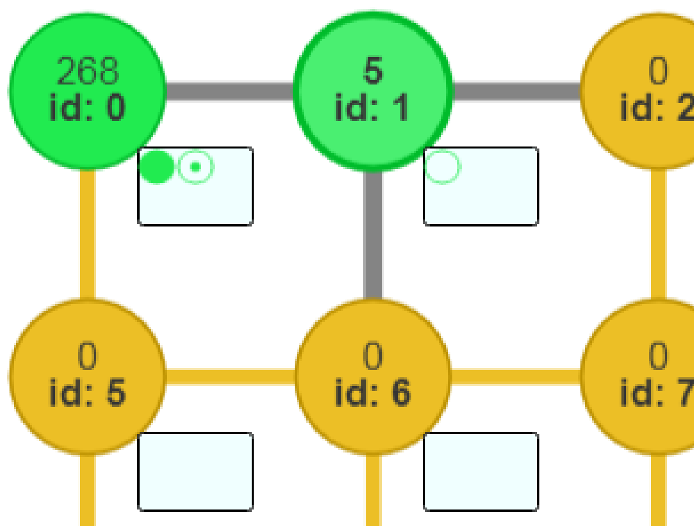
Po přidání agentů se u domovského uzlu objeví kolečka reprezentující agenty. Následně lze na libovolného z nich kliknout, přičemž se objeví ovládací panel agenta. Zde je možné dát agentovi úkol, a to kliknutím na jedno z tlačítek „Drop“, „Take“, „Move“.



Po kliknutím na jedno z tlačítek akcí je následně třeba kliknout na uzel, ve kterém chceme, aby agent akci provedl. Na následujícím obrázku je možné vidět rozložení agentů po několika akcích „Take“ a „Move“. Velikost vnitřního kruhu v kružnici reprezentující agenta je závislá na počtu kreditů, které agent v daný moment nese.

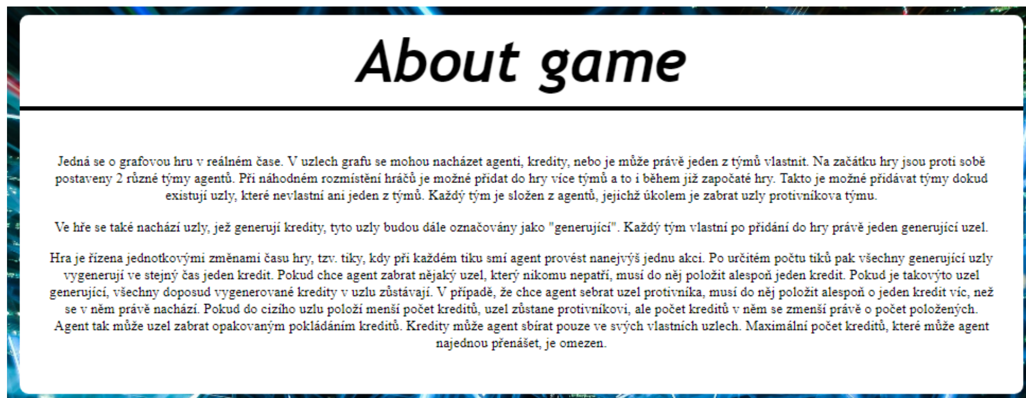


Po vykonání akce „Drop“ ve volném uzlu je tento uzel přivlastněn hráči, jehož agent do tohoto uzlu položil své kredity. To lze vidět na následujícím obrázku v uzlu s identifikačním číslem 1.



Pokud hráč nechce vytvářet novou hru, ale chce se připojit do již existující, klikne v menu na tlačítko „Join Game“. Následně se mu zobrazí seznam aktuálně probíhajících her a po kliknutí na některou z nich je do této hry přidán.

V menu může uživatel zvolit i zobrazení pravidel hry stisknutím tlačítka „Hjälp“.



Vytvoření agenta

Pro vytvoření agenta je třeba založit nový skript v jazyce Python. Tento skript se musí skládat z třídy, která dědí od obecného předka *Agent*. Zároveň musí importovat třídu *Agent* z implementované knihovny *lib.agent*. Vlastní implementace logiky agenta musí být implementována v metodě *run*, která musí být přepsána. Následná ukázka skriptu zobrazuje kód pasivního agenta, který zároveň slouží jako konstra pro implementaci složitějších agentů.

```
from lib.agent import Agent

class PassiveAgent(Agent):

    def run(self):

        while(True):
            game_act = self.get_game_msg()

            if self.instan.game_ended:
                break

            self.enquiry()
```

Pro vyzkoušení implementovaného agenta je možné buď přímo volat jeho metody, nebo je možné ho inicializovat prostřednictvím třídy *Team* z implementované knihovny *lib.team*.

Příklad prvního způsobu je možné vidět v následujícím úryvku kódu:

```
agent = PassiveAgent()
agent.initialize("127.0.0.1", 7890, "team", "game")
agent.start()
agent.wait_for_end()
agent.end()
```

Druhým způsobem lze agenta přidat do týmu prostřednictvím třídy *Team* následujícím způsobem:

```
from lib.team import Team

team = Team("127.0.0.1", 7890, "team", "game")
team.add_agent(PassiveAgent())
team.play()
```

Tento způsob má pak výhodu zejména v možnosti naráz inicializovat více agentů v rámci jednoho týmu. Tento způsob možné uskutečnit pouze v rámci jednoho síťového uzlu.

Jakmile je agent připojen do hry, může se uživatel přihlásit prostřednictvím webového rozhraní do jím vytvořené hry a hrát proti němu.